The Computability Path Ordering

Frédéric Blanqui, Jean-Pierre Jouannaud, Albert Rubio

THE COMPUTABILITY PATH ORDERING

FRÉDÉRIC BLANQUI, JEAN-PIERRE JOUANNAUD, AND ALBERT RUBIO

INRIA, Deducteam, France

École Polytechnique, LIX, and Université Paris-Sud, France

Technical University of Catalonia, Spain

Abstract. This paper aims at carrying out termination proofs for simply typed higher-order calculi automatically by using ordering comparisons. To this end, we introduce the computability path ordering (CPO), a recursive relation on terms obtained by lifting a precedence on function symbols. A first version, core CPO, is essentially obtained from the higher-order recursive path ordering (HORPO) by eliminating type checks from some recursive calls and by incorporating the treatment of bound variables as in the so-called computability closure. The well-foundedness proof shows that core CPO captures the essence of computability arguments *à la* Tait and Girard, therefore explaining its name. We further show that no more type check can be eliminated from its recursive calls without loosing well-foundedness, but one for which we found no counter-example yet. Two extensions of core CPO are then introduced which allow one to consider: the first, higher-order inductive types; the second, a precedence in which some function symbols are smaller than application and abstraction.

1. Introduction

This paper addresses the problem of automating termination proofs for typed higher-order calculi by reducing them to ordering comparisons between lefthand and righthand sides of rules.

It also addresses another, more fundamental problem of mathematical importance. Consider the set of terms generated by a denumerable set of variables, application, abstraction and some set of function symbols with arities, our version of the pure λ-calculus. We shall use a (possibly infinite) set R of pairs of λ-terms called rewrite rules used as our computing device. Given a term as input, whether our computing device will eventually terminate and return an answer is *in general* undecidable, even if R is a singleton set [32]. It may even be undecidable for specific rewrite systems, such as the well-known β-reduction rule (formally defined here as the infinite set of its instances). A major question is the following: can we approximate the set of β-terminating terms by some meaningful subset? An important partial answer was given by Turing: the set of simply-typed λ-terms, where the word *simply*

Frédéric Blanqui thanks the Institute of Software of the Chinese Academy of Sciences for hosting him from June 2012 to August 2013. This research was supported by the Spanish MINECO under grant TIN2013-45732-C4-3-P.

LOGICAL METHODS IN COMPUTER SCIENCE DOI:10.2168/LMCS-???

refers to a specific typing discipline introduced by Church in the λ-calculus [25], is terminating when a specific strategy is employed [89]. The complete answer, the fact that the very same set is indeed terminatin under any strategy, is due to Sanchis [81]. Tait and Girard gave later proofs [84, 45] which have been the basis of many further generalizations, by considering more rules (η-reduction, recursors, general schema), and more terms characterized by more elaborate type disciplines (polymorphic, dependent, inductive type systems). When considering β-reduction alone, the obtained approximations of the set of terminating λ-terms are quite satisfactory. But proving the corresponding statement that computations terminate when given a typed λ-term as input, requires using an extremely powerful technique called *reducibility*[1], introduced by Tait for simply typed terms, and further developed by Girard for the richer type disciplines. Given a set of terms and a set of rewrite rules R, a reducibility predicate is defined by axioms that it should satisfy, mainly closure under term constructions, closure under rewriting with R, and containment in the set of terminating terms. Girard exhibited a particular predicate for β-reduction which can be easily adapted for other sets of rules, but there are sets of rules for which some typable terms originate an infinite computation. We therefore turn to a new undecidable question: which sets of rules admit a computability predicate?

The question we answer in this paper is whether this set S (of sets R of rules) admits some non-trivial decidable subset: our approximation of S is the set of sets R of rules such that pairs in R are ordered by (some instance of) the computability path ordering CPO.

The work itself takes its roots in early attempts by Breazu-Tannen and Gallier [24] and independently Okada [77] to consider mixed typed λ-calculi with algebraic rewriting. Both works used Girard's computability predicates method to show that the strong normalization property of algebraic rewriting was preserved in the union. These results grew into a whole new area, by extending the type discipline on the one hand, and the kind of rules that could be taken care of on the other hand. The type discipline was extended independently by Barbanera and Dougherty in order to cover the whole calculus of constructions [3, 38], while the rule format was extended as described next.

Higher-order rewrite rules satisfying the *general schema*, a generalization of Gödel's primitive recursion rules for higher types, were introduced by Jouannaud and Okada in the case of a polymorphic type discipline [56, 57]. The latter work was then extended first by Barbanera and Fernández [5, 4] and finally by Barbanera, Fernández and Geuvers to cover the whole calculus of constructions [6]. Recursors for *basic* inductive types, which constructors admit arguments of a non-functional type only, could be taken care of by the general schema, but arbitrary strictly positive inductive types could not, prompting for an extension of the schema, which was reformulated for that purpose by Blanqui, Jouannaud and Okada [15]. This new formulation was based on the notion of *computability closure* of a term $\mathsf{f}(\vec{t})$, defined as a set of terms containing $\vec{t}$ and closed under computability preserving operations in the sense of Tait and Girard. Membership to the general schema was then defined for an arbitrary rewrite rule as membership of its righthand side to the computability closure of its lefthand side. This elegant, flexible and powerful definition of the general schema was finally extended by Blanqui in a series of papers, until it covered the entire calculus of inductive constructions including strong elimination rules [10, 11], rewriting modulo some equational theories and rewriting with higher-order pattern-matching [14].

[1] In fact, Tait speaks of "convertibility" in [83], "realizability" in [84]; and Girard of "réductibilité" and "reducibility" in [44, 45, 46]. Following Gödel, "computability" is used by Troelstra in [88], p. 100.

Introduced by Jouannaud and Rubio, HORPO was the next step, the very first order on simply typed λ-terms defined by induction on the term structure, as does Dershowitz recursive path ordering for first-order terms [34]. Comparing two terms with HORPO starts by comparing their types in a given well-founded ordering on types before to proceed recursively on the structure of the compared terms, in a way which depends on a comparison of the roots of both terms in a given well-founded order on the algebraic signature called the precedence [58]. HORPO was extended to the calculus of constructions by Walukiewicz [91], and to use semantic interpretations of terms instead of a precedence on function symbols by Borralleras and Rubio [22]. An axiomatic presentation of the rules underlying HORPO can be found in [47]. A more recent work in the same direction is [35]. A more general version of HORPO appears in [59], which uses the computability closure to strengthen its expressivity. Blanqui proved that the first version of HORPO is contained in an order defined as a fixpoint of the computability closure definition [12]. Indeed, HORPO and the computability closure share many similar constructs, raising expectations for a simpler and yet more expressive definition, instead of a pair of mutually inductive definitions for the computability closure and the ordering itself. On the positive side, the computability closure makes little use of type comparisons, hence may succeed when HORPO fails for type reason. Unfortunately, its fixpoint is not a syntax-oriented definition, hence has a more limited practical usage.

Originally formulated in [16], the question of finding a syntax oriented recursive definition of HORPO that would inherit the advantages of the computability closure paved the way to CPO, the computability path ordering. The first definition was given in [17], later improved as CPO in [18]. A major improvement of CPO is that type comparisons are no more systematic, but occur in very specific cases. This does not only speed up computations, but also boosts the ordering capabilities in an essential way. Further, bound variables are handled explicitly by CPO, allowing for arbitrary abstractions in the righthand sides together with a more uniform definition.

In this paper, we present an in-depth study of an improved version of CPO for a simple extension of Church's simple type discipline [25], before we extend it to inductive types along the lines suggested in [18] following a technique dating back to Mendler [73, 74] and extended to rewriting by Blanqui [10]. In particular, we first show that many improvements of CPO cannot be well-founded: type comparisons are necessary when recursive calls deconstruct the lefthand side, but are not otherwise. While this all came out of the well-foundedness proof, it indeed shows a strong relationship between the recursive structure of CPO and the computability predicates method of Tait and Girard that is used to carry out the proof, which explains the name CPO. It further shows that CPO is indeed a sharp approximation of the set of sets of rules which admit a computability predicate. We then address the treatment of inductive types which remained *ad hoc* so far, thanks to the use of accessibility, a relationship introduced by Blanqui which generalizes the notion of inductive type [11]. We finally introduce another novelty: small symbols. In all previous definitions, function symbols were bigger in the precedence than application and abstraction. Such symbols are now called *big*, while *small* symbols behave differently, being possibly smaller than both. Small symbols were suggested by Li to carry out a generalization of CPO to dependent types [55].

In the recent years, the success of HORPO has prompted interest in the generalization to higher-order computations of various other methods used for first-order computations, most notably Art and Giesl's dependency pairs [2, 42, 50] yielding for instance [80, 68, 64, 82, 62, 63], and interpretation methods [70, 69, 93, 27] yielding for instance [90, 48, 78, 40].

The paper is organized as follows. First, we define the sets of types and terms that we consider (simply typed λ-terms with function symbols of fixed arity), and the class of orderings on types that can be used in CPO. We then give a first definition of our ordering (core CPO), and show that it can hardly be improved while keeping the same recursive structure and well-foundedness. We then show how to prove its well-foundedness by extending Tait and Girard's technique of computability predicates. In the following sections, we consider two extensions of core CPO. In the first one, core CPO is extended by using accessible subterms which allows to handle strictly inductive types. In the second, application or abstraction are allowed to be bigger than a function call. Concluding remarks are given in Section 9.

We recommend surveys [36, 86] for rewriting and [7] for typed λ-calculus.

2. Types and admissible type orderings

CPO is a relation on well typed terms but, instead of allowing the comparison of terms of the same type only, it allows the type to decrease in some well-founded ordering. However, not any type ordering is admissible.

In this section, we first recall the definition of (simple) *types* and some basic functions on types. Then, we define what are the (strict) orderings on types that can be used in CPO, study some of their properties and give an example based on a well-founded precedence on type constants.

Definition 2.1 (Types)**.** Let $\mathcal{S}$ be a set of *sorts*. The set $\mathcal{T}$ of *types*, the *arity* $\alpha(\_)$ and the *order* $o(\_)$ of a type are inductively defined as follows:

- a sort $\mathsf{A} \in \mathcal{S}$ is a type of arity $\alpha(\mathsf{A}) = 0$ and order $o(\mathsf{A}) = 0$.
- if T and U are types, then $T \to U$ is a type of arity $\alpha(T \to U) = 1 + \alpha(U)$ and order $o(T \to U) = \max\{1 + o(T), o(U)\}$.

We use capital letters for types and a different font for sorts (*e.g.* T and A), and $\vec{T}$ for a (possibly empty) sequence of types $T_1, \dots, T_n$, of *length* $|\vec{T}| = n$.

As usual, $\to$ associates to the right so that $\mathsf{A} \to \mathsf{A} \to \mathsf{A}$ and $\mathsf{A} \to (\mathsf{A} \to \mathsf{A})$ are the same.

Given a relation R, let R^+ (resp. R^*) denote the transitive (resp. transitive and reflexive) closure of R.

Definition 2.2 (Admissible type orderings)**.** Let $\triangleright_l$ and $\triangleright_r$ be the relations on types such that $T \to U \ \triangleright_l \ T$ and $T \to U \ \triangleright_r \ U$ respectively, and $\triangleright$ be the transitive closure of their union. A (strict) ordering $>$ on types is *admissible* if:

- $\triangleright_r \subseteq >$ **(typ-right-subterm)**
- $\gtrdot = (> \cup \triangleright_l)^+$ is well-founded **(typ-sn)**
- if $T \to U > V$, then $U \geq V$ or $V = T \to U'$ with $U > U'$ **(typ-arrow)**

where $\geq$ is the reflexive closure of $>$. We say that a type T is compatible (resp. strictly compatible) with a sort A, written $\mathrm{Sort}_{\leq \mathsf{A}}(T)$ (resp. $\mathrm{Sort}_{< \mathsf{A}}(T)$) if $\mathsf{B} \leqslant \mathsf{A}$ (resp. $\mathsf{B} \lessdot \mathsf{A}$) for every sort B occurring in T.

Admissible type orderings originate from [59]. Note that a sort can be bigger than an arrow type. If A is a sort occurring in T, then $T \geqslant \mathsf{A}$. Finally, note that the relation $\gtrdot$ is a *simplification ordering* [34].

We now give an example of admissible ordering based on a well-founded precedence on sorts. For a concrete use case, see Example 5.2 below.

Lemma 2.3. *Given a well-founded ordering $>_{\mathcal{S}}$ on sorts, let $>$ be the smallest ordering $>$ on types containing $>_{\mathcal{S}}$ and $\triangleright_r$ and such that, for all U, V, V', it holds that $V > V'$ implies $U \to V > U \to V'$. Then, $>$ is admissible.*

Proof.

- (typ-sn) $\triangleright$ is included in the RPO extending $>_{\mathcal{S}}$ [33], hence is well-founded.
- (typ-right-subterm) By definition.
- (typ-arrow) Let $T \to U > V$. The proof is by induction on the definition of $>$.
  (1) $>$ is $>_{\mathcal{S}}$. Impossible since $T \to U$ is not a sort.
  (2) $>$ is $\triangleright_r$. Then $U = V$, hence $U \geq V$.
  (3) $V = T \to W$ and $U > W$. Immediate.
  (4) $T \to U > W > V$. By induction hypothesis applied to $T \to U > W$, there are two cases:
    - $U \geq W$. Then, by transitivity, $U > V$.
    - $W = T \to U'$ for some $U' < U$. By induction hypothesis on $T \to U' > V$, there are two cases:
      * $U' \geq V$. By transitivity, $U > V$.
      * $V = T \to V'$ for some $V' < U'$. By transitivity, $U > V'$ and we are done. □

In the following, we prove some properties of admissible type orderings:

Lemma 2.4. *Let $>$ be an admissible type ordering. If $T \to U > T' \to U'$, then $U > U'$.*

Proof. By (typ-arrow), either $T = T'$ and $U > U'$ or $U \geq T' \to U'$, in which case we conclude by (typ-right-subterm) and transitivity. □

Lemma 2.5. *Let $>$ be an admissible type ordering. If $\mathsf{A} > U$, then $\mathrm{Sort}_{\triangleleft \mathsf{A}}(U)$.*

Proof. Let B be a sort occurring in U. Then, $U \geq \mathsf{B}$. Hence, by transitivity, $\mathsf{A} \triangleright \mathsf{B}$. □

Lemma 2.6. *Let $>$ be an admissible type ordering. If $T > U$ and $\mathrm{Sort}_{\trianglelefteq \mathsf{A}}(T)$, then $\mathrm{Sort}_{\trianglelefteq \mathsf{A}}(U)$.*

Proof. Let C be a sort occurring in U. We proceed by induction on T.

- $T = \mathsf{B}$. Since $\mathrm{Sort}_{\trianglelefteq \mathsf{A}}(T)$, $\mathsf{B} \trianglelefteq \mathsf{A}$. By Lemma 2.5, $\mathrm{Sort}_{\triangleleft \mathsf{B}}(U)$ and $C \triangleleft \mathsf{B}$. Therefore, by transitivity, $C \trianglelefteq \mathsf{A}$.
- $T = S \to T'$. Then, $\mathrm{Sort}_{\trianglelefteq \mathsf{A}}(S)$ and $\mathrm{Sort}_{\trianglelefteq \mathsf{A}}(T')$. By (typ-arrow), there are two cases:
  - $T' \geq U$. Then, by induction hypothesis, $\mathrm{Sort}_{\trianglelefteq \mathsf{A}}(U)$.
  - $U = S \to U'$ and $T' > U'$. By induction hypothesis, $\mathrm{Sort}_{\trianglelefteq \mathsf{A}}(U')$. Hence $\mathrm{Sort}_{\trianglelefteq \mathsf{A}}(U)$. □

3. Terms

In this section, we define the set of terms on which CPO operates. We consider simply-typed λ-terms [25, 7] with function symbols of fixed arity, that is, a function symbol of arity n always comes with n arguments. We assume that every variable or function symbol comes equipped with a fixed type and that α-equivalence replaces a variable by another variable of the same type.

Definition 3.1 (Terms)**.** Let $\mathcal{X}$ be an infinite set of *variables*, each variable x being equipped with a type $\tau(x) \in \mathcal{T}$ so that there is an infinite number of variables of each type. Let also $\mathcal{F}$ be a (finite or infinite) set of *function symbols* disjoint from $\mathcal{X}$, each function symbol f being equipped with a type $\tau(\mathsf{f}) \in \mathcal{T}$ and an arity $\alpha(\mathsf{f}) \leq \alpha(\tau(\mathsf{f}))$. The *declaration* $\mathsf{f}^n : T$ indicates the arity n and type T of f. The set $\mathcal{L}$ of terms is defined inductively as follows:

- a variable x is a term of type $\tau(x)$;
- if $\mathsf{f}^n : T_1 \to \ldots \to T_n \to U$ and $t_1, \ldots, t_n$ are terms of type $T_1, \ldots, T_n$ respectively, then $\mathsf{f}(t_1, \ldots, t_n)$ is a term of type U;
- if t and u are terms of types $U \to V$ and U respectively, then tu is a term of type V;
- if x is a variable and t is a term of type T, then λxt is a term of type $\tau(x) \to T$.

We denote by $\tau(t)$ the type of a term t, and write $t : T$ when $\tau(t) = T$.

We usually write $\mathsf{f} : T$ for the declaration $\mathsf{f}^0 : T$, omitting the arity $n = 0$, and f for $\mathsf{f}()$. Note that a term $\mathsf{f}(\vec{t})$ may have a functional type, hence can be applied. Application associates to the left so that tuv is the same as $(tu)v$.

We use the letters $x, y, z, \ldots$ for variables, $\mathsf{f}, \mathsf{g}, \ldots$ for function symbols, and $a, b, \ldots$, $s, t, u, v, \ldots$, $t', u', \ldots$ for terms.

We denote by $\mathrm{FV}(t)$ the set of free variables in t, by $\lhd$ the strict subterm relationship on terms, and by $\trianglelefteq$ its reflexive closure. The height of a term t, written $|t|$, is the height of its tree representation: $|x| = 0$, $|\mathsf{f}| = 0$ if $\alpha(\mathsf{f}) = 0$, $|\mathsf{f}(\vec{t})| = 1 + \max\{|t_i| \mid 1 \leq i \leq \alpha(\mathsf{f})\}$ if $\alpha(\mathsf{f}) > 0$, $|tu| = 1 + \max\{|t|, |u|\}$ and $|\lambda xt| = 1 + |t|$.

Definition 3.2 (Substitution)**.**

- A substitution is a function $\sigma : \mathcal{X} \to \mathcal{L}$ such that $\mathrm{dom}(\sigma) = \{x \in \mathcal{X} \mid \sigma(x) \neq x\}$ is finite and, for every x, $\tau(\sigma(x)) = \tau(x)$. As usual, the application of a substitution σ to a term t, written $t\sigma$, is defined so as to avoid free-variable captures when renaming some bound variables of t by new variables of the same type [31].
- A substitution σ is *away from* a finite set of variables X if $(\mathrm{dom}(\sigma) \cup \mathrm{FV}(\sigma)) \cap X = \varnothing$, where $\mathrm{FV}(\sigma) = \bigcup\{\mathrm{FV}(\sigma(x)) \mid x \in \mathrm{dom}(\sigma)\}$.
- A relation $>$ on terms (or sequences of terms) is stable by substitution away from X if $a\sigma > b\sigma$ whenever $a > b$ and σ is away from X. A relation is stable by substitution if it is stable by substitution away from $\varnothing$.

We will use the letters $\sigma, \theta, \ldots$ for substitutions, and denote the substitution mapping the variables $\vec{x}$ of its domain to the terms in $\vec{t}$ (hence $|\vec{x}| = |\vec{t}|$) by $\binom{\vec{t}}{\vec{x}}$.

Note that stability by substitution reduces to the standard definition: $>$ is stable by substitution if $a\sigma > b\sigma$ whenever $a > b$ (because any substitution is away from $\varnothing$).

The equivalence relation identifying terms up to type-preserving renaming of their bound variables is called α-equivalence and written $=_\alpha$ as usual [31].

Given a relation R, let $\mathrm{SN}_T(R)$ be the set of terms of type T from which there is no infinite sequence of R-steps, and $\mathrm{SN}(R) = \bigcup\{\mathrm{SN}_T(R) \mid T \in \mathcal{T}\}$.

4. Relations

One ingredient of CPO is a well-founded quasi-ordering on function symbols and, for each equivalence class generated by the corresponding equivalence relation, a *status* $\mathrm{stat} \in \{\mathrm{mul}\} \cup \{\mathrm{lex}(n) \mid n > 2\}$ prescribing how to compare the arguments of two equivalent symbols, by either its multiset [37] or lexicographic extension. We hereafter recall the necessary

definitions and state some simple but important properties of these operations. The product extension is introduced here for technical reasons.

Given a relation $\succ$ on terms, let:

- $\vec{t} \succ_{\text{prod}} \vec{u}$ if $|\vec{t}| = |\vec{u}|$ and there is $j \in \{1, \dots, |\vec{u}|\}$ s.t. $t_j \succ u_j$ and, for all $i \neq j$, $t_i = u_i$.
- $\vec{t} \succ_{\text{mul}} \vec{u}$ if $\{\!\{\vec{t}\}\!\} (\succ^1_{\mathcal{M}})^+ \{\!\{\vec{u}\}\!\}$ where $\{\!\{\vec{t}\}\!\}$ is the multiset made of the elements in $\vec{t}$ and $M + \{\!\{x\}\!\} \succ^1_{\mathcal{M}} M + \{\!\{y_1, \dots, y_n\}\!\}$ $(n \geq 0)$ if, for all i, $x \succ y_i$ (+ being the multiset union);
- $\vec{t} \succ_{\text{lex}(n)} \vec{u}$ if there is $j \in \{1, \dots, n\}$ such that $t_j \succ u_j$ and, for all $i < j$, $t_i = u_i$.

Note that both $\succ_{\text{mul}}$ and $\succ_{\text{lex}(n)}$ may compare all the arguments whatever their types are (from left to right for $\succ_{\text{lex}(n)}$). In [14], the first author describes a more general version of these statuses that take types into account and allow reordering and filtering of the arguments [2]. We could also consider statuses combining both lexicographic and multiset comparisons [39].

In the following, we will omit n in $\succ_{\text{lex}(n)}$ and simply write $\succ_{\text{lex}}$.

Here are the properties of statuses we will rely on:

Proposition 4.1. *Given a relation $\succ$ on terms:*

- $\succ_{\text{stat}}$ *preserves termination: if $\succ$ is well-founded, then $\succ_{\text{stat}}$ is well-founded.*
- $\succ_{\text{stat}}$ *contains* $\succ_{\text{prod}}$.
- $\succ_{\text{stat}}$ *preserves stability: if $\succ$ is stable by substitution away from X, then so is $\succ_{\text{stat}}$.*

5. Computability path ordering

In this section, we give the core definition of the computability path ordering (CPO) before to explore its limits by means of examples and compare it with its father definition, HORPO.

5.1. Definition of core CPO.

We assume given:

- an admissible ordering on types $>$;
- a quasi-ordering $\geq_{\mathcal{F}}$ on $\mathcal{F}$, called *precedence*, whose equivalence $\geq_{\mathcal{F}} \cap \geq_{\mathcal{F}}^{-1}$ is written $\simeq_{\mathcal{F}}$ and strict part $\geq_{\mathcal{F}} \setminus \geq_{\mathcal{F}}^{-1}$ is written $>_{\mathcal{F}}$ and assumed well-founded;
- for every $\mathsf{f} \in \mathcal{F}$, a status $\text{stat}(\mathsf{f}) \in \{\text{mul}\} \cup \{\text{lex}(n) \mid n \geq 2\}$ such that symbols equivalent in $\simeq_{\mathcal{F}}$ have the same status.

Definition 5.1 (Core computability path relation)**.** The core computability path relation is the relation $\succ_\tau^\varnothing$ ($\succ_\tau$ for short) where:

- the set $\mathcal{F}_b$ of *big* symbols is identical to $\mathcal{F}$,[2]
- for any given finite set X of variables, $\succ^X$ is inductively defined in Figure 1,
- $t \succ_\tau^X u$ if $t \succ^X u$ and $\tau(t) \geq \tau(u)$,
- $\geq^X$ (resp. $\geq_\tau^X$) is the reflexive closure of $\succ^X$ (resp. $\succ_\tau^X$).

The parameter X serves as a meta-level binder to keep track of the variables that were previously bound in the righthand side but have become free when destructuring a righthand side abstraction. We shall say that a variable x is *fresh* with respect to a comparison $u \succ^X v$ if $x \notin \text{FV}(u) \cup X \cup \text{FV}(v)$.

[2] In core CPO, all symbols are big. *Small* symbols will show up in Section 8.

Figure 1: Core CPO

$$
\begin{array}{rl}
(\mathcal{F}_b\triangleright) & \mathsf{f}(\vec{t}) \succ^X v \text{ if } \mathsf{f} \in \mathcal{F}_b \text{ and } (\exists i)\ t_i \succeq_\tau v \\
(\mathcal{F}_b{=}) & \mathsf{f}(\vec{t}) \succ^X \mathsf{g}(\vec{u}) \text{ if } \mathsf{f} \in \mathcal{F}_b,\ \mathsf{f} \simeq_{\mathcal{F}} \mathsf{g},\ (\forall i)\ \mathsf{f}(\vec{t}) \succ^X u_i \text{ and } \vec{t}\,(\succ_\tau)_{\mathrm{stat}(\mathsf{f})}\,\vec{u} \\
(\mathcal{F}_b{>}) & \mathsf{f}(\vec{t}) \succ^X \mathsf{g}(\vec{u}) \text{ if } \mathsf{f} \in \mathcal{F}_b,\ \mathsf{f} \succ_{\mathcal{F}} \mathsf{g} \text{ and } (\forall i)\ \mathsf{f}(\vec{t}) \succ^X u_i \\
(\mathcal{F}_b@) & \mathsf{f}(\vec{t}) \succ^X uv \text{ if } \mathsf{f} \in \mathcal{F}_b,\ \mathsf{f}(\vec{t}) \succ^X u \text{ and } \mathsf{f}(\vec{t}) \succ^X v \\
(\mathcal{F}_b\lambda) & \mathsf{f}(\vec{t}) \succ^X \lambda yv \text{ if } \mathsf{f} \in \mathcal{F}_b,\ \mathsf{f}(\vec{t}) \succ^{X\cup\{z\}} v_y^z,\ \tau(y) = \tau(z) \text{ and } z \text{ fresh} \\
(\mathcal{F}_b\mathcal{X}) & \mathsf{f}(\vec{t}) \succ^X y \text{ if } \mathsf{f} \in \mathcal{F}_b \text{ and } y \in X \\
(@\triangleright) & tu \succ^X v \text{ if } t \succeq^X v \text{ or } u \succeq_\tau^X v \\
(@{=}) & tu \succ^X t'u' \text{ if } t = t' \text{ and } u \succ^X u', \text{ or } tu \succ_@^X t' \text{ and } tu \succ_@^X u' \\
 & \text{where } tu \succ_@^X v \text{ if } t \succ_\tau^X v \text{ or } u \succeq_\tau^X v \text{ or } tu \succ_\tau^X v \\
(@\lambda) & tu \succ^X \lambda yv \text{ if } tu \succ^X v_y^z \text{ and } z \text{ fresh} \\
(@\mathcal{X}) & tu \succ^X y \text{ if } y \in X \\
(@\beta) & (\lambda xt)u \succ^X v \text{ if } t_x^u \succeq^X v \\
(\lambda\triangleright) & \lambda xt \succ^X v \text{ if } t_x^z \succeq_\tau^X v,\ \tau(x) = \tau(z) \text{ and } z \text{ fresh} \\
(\lambda{=}) & \lambda xt \succ^X \lambda yv \text{ if } t_x^z \succ^X v_y^z,\ \tau(x) = \tau(y) = \tau(z) \text{ and } z \text{ fresh} \\
(\lambda{\neq}) & \lambda xt \succ^X \lambda yv \text{ if } \lambda xt \succ^X v_y^z,\ \tau(x) \neq \tau(y),\ \tau(y) = \tau(z) \text{ and } z \text{ fresh} \\
(\lambda\mathcal{X}) & \lambda xt \succ^X y \text{ if } y \in X \\
(\lambda\eta) & \lambda x(tx) \succ^X v \text{ if } t \succeq^X v \text{ and } x \notin \mathrm{FV}(t)
\end{array}
$$

Note that the parameter X is carried along computations without change, except in rule $(\mathcal{F}_b\lambda)$. Hence, any comparison $u \succ^X v$ generated from an initial comparison $s \succ^\varnothing t$ implies $X \cap \mathrm{FV}(u) = \varnothing$.

Explicit variable renamings and the associated freshness conditions are used to make the relation invariant by α-equivalence, the smallest congruence generated by the equation $\lambda xt = \lambda yt_x^y$ if $\tau(x) = \tau(y)$ and $y \notin \mathrm{FV}(\lambda xt)$ [31], and by appropriate renaming of the variables in X, as we shall prove later.

Note the seemingly complex behaviour of application in rule (@=), which allows to search the lefthand side for appropriate arguments bigger than those of the righthand side. This enhancement of CPO intends to mimic the corresponding rule of HORPO without flattening lefthand sides.

Having function symbols equipped with an arity is more general than having uncurried function symbols (*i.e.* of null arity) only: any uncurried system can be dealt with as it is. However, in this case, the $(\mathcal{F}_b\_)$ rules are very limited: $(\mathcal{F}_b\triangleright)$ is not applicable, $(\mathcal{F}_b{=})$ and $(\mathcal{F}_b{>})$ reduce to the precedence itself. Moreover, applications of the form $\mathsf{f}\vec{t}$ with $|\vec{t}| > 0$ can only be compared by using the (@_) rules which are more constrained than the corresponding $(\mathcal{F}_b\_)$ rules, especially $(@\lambda)$ and (@=). Considering function symbols with non-null arities provides more structure to the terms, and this structure can be used for proving termination [51].

Lemma 6.4 below will show that $\mathrm{FV}(v) \subseteq \mathrm{FV}(u) \cup X$ whenever $u \succ^X v$. Hence, an alternative formulation of rules $(@\lambda)$ and $(\lambda{\neq})$ could therefore be given by replacing the condition "z fresh" by $y \notin \mathrm{FV}(v)$.

Another, perhaps surprising fact is that the definition of core CPO can be simplified by replacing $\succ^X$ by $\succ$ everywhere but in $(\mathcal{F}_b\lambda)$. This is true at the start since we are interested

in $\succ_\tau$. This is then an invariant of the computation, for two reasons: X is never increased, except in $(\mathcal{F}_b\lambda)$; X is reset to the empty set by $(\mathcal{F}_b\triangleright)$ and $(\mathcal{F}_b{=})$, which are the only rules which may move from a $(\mathcal{F}_b)$ comparison to a $(@)$ or (λ) comparison. We could therefore simplify our definition by removing the superfluous X subscripts. This will however no more be true of the extension of core CPO to inductive types, and we prefer to have a uniform definition over the various sections. Further, the present definition will allow us to study a relaxation of $(@\lambda)$ in the next section.

Surprisingly, core CPO is powerful enough already to prove termination of examples that usually require techniques like the ones developed in Section 7.

Example 5.2. Consider the breadth-first search of labeled trees using continuations [52], using the sorts L for lists of labels and C for continuations, the abbreviation $\neg T = T \to \mathsf{L}$, and the symbols $\mathsf{d} : \mathsf{C}$ and $\mathsf{c}^1 : \neg\neg\mathsf{C} \to \mathsf{C}$ for building continuations. Let now $\mathsf{e} : \neg\mathsf{C}$ defined by the rule:

$$\mathsf{e}\,\mathsf{c}(x) \to x\,\mathsf{e}$$

Its termination can be checked by core CPO by taking $\mathsf{C} \geq \mathsf{L}$ and $\mathsf{c} >_\mathcal{F} \mathsf{e}$. Indeed, $\mathsf{e}\,\mathsf{c}(x) \succ_\tau x\,\mathsf{e}$ holds by $(@\triangleright)$ since $\mathsf{c}(x) \succ_\tau x\,\mathsf{e}$ for $\tau(\mathsf{c}(x)) = \mathsf{C} \geq \tau(x\,\mathsf{e}) = \mathsf{L}$ and, by $(\mathcal{F}_b@)$, $\mathsf{c}(x) \succ x$ by $(\mathcal{F}_b\triangleright)$, and $\mathsf{c}(x) \succ \mathsf{e}$ by $(\mathcal{F}_b{>})$.

5.2. **Transitivity.** As HORPO, core CPO is not transitive (both include β-reduction which is not transitive). Adding transitivity as a rule yields non-termination as shown by the following counter-example:

Example 5.3. In the premises of $(\mathcal{F}_b@)$, replace $\succ^X$ by $(\succ^X)^+$. Then, in the system S1 described at the beginning of next section, we have:

(1) $\mathsf{f}(\mathsf{a}) \succ_\tau (\lambda x\mathsf{f}(x))\mathsf{a}$ since $\tau(\mathsf{f}(\mathsf{a})) = o \geq \tau((\lambda x\mathsf{f}(x))\mathsf{a}) = o$ and, by relaxing $(\mathcal{F}_b@)$:
  (a) $\mathsf{f}(\mathsf{a}) \succ^+ \lambda x\mathsf{f}(x)$ since
    (i) $\mathsf{f}(\mathsf{a}) \succ \mathsf{a}$ by $(\mathcal{F}_b\triangleright)$, and
    (ii) $\mathsf{a} \succ \lambda x\mathsf{f}(x)$ by $(\mathcal{F}_b\lambda)$ since $\mathsf{a} \succ^{\{x\}} \mathsf{f}(x)$ by $(\mathcal{F}_b{>})$ and then $(\mathcal{F}_b\mathcal{X})$
  (b) $\mathsf{f}(\mathsf{a}) \succ \mathsf{a}$ by $(\mathcal{F}_b\triangleright)$.
(2) $(\lambda x\mathsf{f}(x))\mathsf{a} \succ_\tau \mathsf{f}(\mathsf{a})$ since $\tau((\lambda x\mathsf{f}(x))\mathsf{a}) = o \geq \tau(\mathsf{f}(\mathsf{a})) = o$ and by $(@\beta)$.

Similar counter-examples can be built as well for $(\mathcal{F}_b{>})$ and $(\mathcal{F}_b{=})$, since the key point is that, by using $(\succ^X)^+$, we can apply case $(\mathcal{F}_b\triangleright)$ without requiring type decreasingness.

Useful implemented heuristics for under-approximating $\succ_\tau^+$ are discussed in [59]. The introduction of small symbols in Section 8 will reduce the need for such heuristics, although not completely. On the other hand, we will show soon that core CPO is a well-founded relation on terms. So is therefore its transitive closure.

5.3. **Tightness of core CPO.** In this section, we show that almost all possible relaxations (by replacing $\succ_\tau$ by $\succ$, and $\succ$ by $\succ^X$) of the above definition lead to non-termination by providing appropriate examples that are also meant to help understanding how CPO works. To this end, we will consider three different systems, using $o : *$ to declare a sort o:

S1: $o : *;\ \mathsf{a} : o,\ \mathsf{f}^1 : o \to o,\ \mathsf{g} : o \to o \to o;\ \mathsf{a} >_\mathcal{F} \mathsf{f} >_\mathcal{F} \mathsf{g}$.
S2: $o, o' : *;\ \mathsf{a} : o,\ \mathsf{f}^1 : o \to o,\ \mathsf{j}^1 : (o' \to o \to o) \to o;\ \mathsf{a} >_\mathcal{F} \mathsf{f} >_\mathcal{F} \mathsf{j}$.
S3: $o : *;\ \mathsf{a} : o,\ \mathsf{h}^2 : o \to o \to o,\ \mathsf{k}^2 : (o \to o) \to o \to o;\ \mathsf{a} >_\mathcal{F} \mathsf{h} \simeq_\mathcal{F} \mathsf{k};\ \mathrm{stat}(\mathsf{h}) = \mathrm{stat}(\mathsf{k}) = \mathrm{mul}$.

For each rule, we now consider all its natural relaxations.

- $(\mathcal{F}_b\triangleright)$ $\mathsf{f}(\vec{t}) \succ^X v$ if $(\exists i) t_i \succeq_\tau v$
  - Replace $\succeq_\tau$ by $\succeq_\tau^X$. Then, in S1, we have:
    (1) $\mathsf{f}(\mathsf{a}) \succ_\tau (\lambda x \mathsf{f}(x))\mathsf{a}$ since $\tau(\mathsf{f}(\mathsf{a})) = o \geq \tau((\lambda x \mathsf{f}(x))\mathsf{a}) = o$ and, by $(\mathcal{F}_b@)$:
      (a) $\mathsf{f}(\mathsf{a}) \succ \lambda x \mathsf{f}(x)$ since, by $(\mathcal{F}_b\lambda)$, $x \notin \mathrm{FV}(\mathsf{f}(\mathsf{a}))$ and
        (i) $\mathsf{f}(\mathsf{a}) \succ^{\{x\}} \mathsf{f}(x)$ since, by relaxing $(\mathcal{F}_b\triangleright)$:
          $\mathsf{a} \succeq_\tau^{\{x\}} \mathsf{f}(x)$ since $\tau(\mathsf{a}) = o \geq \tau(\mathsf{f}(x)) = o$ and, by $(\mathcal{F}_b>)$, $\mathsf{a} \succ^{\{x\}} x$ by $(\mathcal{F}_b\mathcal{X})$,
      (b) $\mathsf{f}(\mathsf{a}) \succ \mathsf{a}$ by $(\mathcal{F}_b\triangleright)$.
    (2) $(\lambda x \mathsf{f}(x))\mathsf{a} \succ_\tau \mathsf{f}(\mathsf{a})$ since $\tau((\lambda x \mathsf{f}(x))\mathsf{a}) = o \geq \tau(\mathsf{f}(\mathsf{a})) = o$ and by $(@\beta)$.
  - Replace $\succeq_\tau$ by $\succeq$. Then, in S1, we have:
    (1) $\mathsf{f}(\mathsf{a}) \succ_\tau (\lambda x \mathsf{f}(x))\mathsf{a}$ since $\tau(\mathsf{f}(\mathsf{a})) = o \geq \tau((\lambda x \mathsf{f}(x))\mathsf{a}) = o$ and, by $(\mathcal{F}_b@)$:
      (a) $\mathsf{f}(\mathsf{a}) \succ \lambda x \mathsf{f}(x)$ since, by relaxing $(\mathcal{F}_b\triangleright)$:
        (i) $\mathsf{a} \succeq \lambda x \mathsf{f}(x)$ since, by $(\mathcal{F}_b\lambda)$, $x \notin \mathrm{FV}(\mathsf{a})$ and
          $\mathsf{a} \succ^{\{x\}} \mathsf{f}(x)$ and, by $(\mathcal{F}_b>)$, $\mathsf{a} \succ^{\{x\}} x$ by $(\mathcal{F}_b\mathcal{X})$
      (b) $\mathsf{f}(\mathsf{a}) \succ \mathsf{a}$ by $(\mathcal{F}_b\triangleright)$.
    (2) $(\lambda x \mathsf{f}(x))\mathsf{a} \succ_\tau \mathsf{f}(\mathsf{a})$ since $\tau((\lambda x \mathsf{f}(x))\mathsf{a}) = o \geq \tau(\mathsf{f}(\mathsf{a})) = o$ and by $(@\beta)$.
- $(\mathcal{F}_b=)$ $\mathsf{f}(\vec{t}) \succ^X \mathsf{g}(\vec{u})$ if $\mathsf{f} \simeq_\mathcal{F} \mathsf{g}$, $\mathsf{f}(\vec{t}) \succ^X \vec{u}$ and $\vec{t}\,(\succ_\tau)_{\mathrm{stat}(\mathsf{f})}\,\vec{u}$
  - Replace $\succ_\tau$ by $\succ_\tau^X$. Then, in S1, we have:
    (1) $\mathsf{f}(\mathsf{a}) \succ_\tau (\lambda x \mathsf{f}(x))\mathsf{a}$ since $\tau(\mathsf{f}(\mathsf{a})) = o \geq \tau((\lambda x \mathsf{f}(x))\mathsf{a}) = o$ and, by $(\mathcal{F}_b@)$:
      (a) $\mathsf{f}(\mathsf{a}) \succ \lambda x \mathsf{f}(x)$ since, by $(\mathcal{F}_b\lambda)$, $x \notin \mathrm{FV}(\mathsf{f}(\mathsf{a}))$ and:
        (i) $\mathsf{f}(\mathsf{a}) \succ^{\{x\}} \mathsf{f}(x)$ since, by relaxing $(\mathcal{F}_b=)$:
          $\mathsf{f}(\mathsf{a}) \succ^{\{x\}} x$ by $(\mathcal{F}_b\mathcal{X})$,
          $\mathsf{a} \succ_\tau^{\{x\}} x$ since $\tau(\mathsf{a}) = o \geq \tau(x) = o$ and by $(\mathcal{F}_b\mathcal{X})$,
      (b) $\mathsf{f}(\mathsf{a}) \succ \mathsf{a}$ by $(\mathcal{F}_b\triangleright)$,
    (2) $(\lambda x \mathsf{f}(x))\mathsf{a} \succ_\tau \mathsf{f}(\mathsf{a})$ since $\tau((\lambda x \mathsf{f}(x))\mathsf{a}) = o \geq \tau(\mathsf{f}(\mathsf{a})) = o$ and by $(@\beta)$.
  - Replace $\succ_\tau$ by $\succ$. We found no counter-example for this case, but this is due to the condition $\mathsf{f}(\vec{t}) \succ^X \vec{u}$. If we consider $(\mathcal{F}_b$=mul) and $(\mathcal{F}_b$=lex) instead, then simple counter-examples like the following one in S3 come up.
    (1) $\mathsf{h}(\mathsf{a},\mathsf{a}) \succ_\tau \mathsf{k}(\lambda x \mathsf{h}(x,x))\mathsf{a}$ since $\tau(\mathsf{h}(\mathsf{a},\mathsf{a})) = o \geq \tau(\mathsf{k}(\lambda x \mathsf{h}(x,x))\mathsf{a}) = o$ and, by relaxing $(\mathcal{F}_b$=mul), $\{\mathsf{a},\mathsf{a}\}\,(\succ)_{\mathrm{mul}}\,\{\lambda x \mathsf{h}(x,x), \mathsf{a}\}$, since $\mathsf{a} \succ \lambda x \mathsf{h}(x,x)$, by case $(\mathcal{F}_b\lambda)$ because $\mathsf{a} \succ^{\{x\}} x$ by case $(\mathcal{F}_b\mathcal{X})$.
    (2) $\mathsf{k}(\lambda x \mathsf{h}(x,x))\mathsf{a} \succ_\tau (\lambda x \mathsf{h}(x,x))\mathsf{a}$ since $\tau(\mathsf{k}(\lambda x \mathsf{h}(x,x))\mathsf{a}) = o \geq \tau((\lambda x \mathsf{h}(x,x))\mathsf{a}) = o$ and by case $(\mathcal{F}_b@)$, since
      (a) $\mathsf{k}(\lambda x \mathsf{h}(x,x))\mathsf{a} \succ \lambda x \mathsf{h}(x,x)$ by $(\mathcal{F}_b\triangleright)$.
      (b) $\mathsf{k}(\lambda x \mathsf{h}(x,x))\mathsf{a} \succ \mathsf{a}$ by $(\mathcal{F}_b\triangleright)$.
    (3) $(\lambda x \mathsf{h}(x,x))\mathsf{a} \succ_\tau \mathsf{h}(\mathsf{a},\mathsf{a})$ since $\tau((\lambda x \mathsf{h}(x,x))\mathsf{a}) = o \geq \tau(\mathsf{h}(\mathsf{a},\mathsf{a})) = o$ and by $(@\beta)$.

    Note that this counter-example can be also applied on case $(\mathcal{F}_b$=lex) if we take $\mathrm{stat}(\mathsf{h}) = \mathrm{stat}(\mathsf{k}) = \mathrm{lex}$. Unfortunately it does not work on $(\mathcal{F}_b=)$ since we cannot prove $\mathsf{h}(\mathsf{a},\mathsf{a}) \succ \lambda x \mathsf{h}(x,x)$.
- $(@\triangleright)$ $tu \succ^X v$ if $t \succeq^X v$ or $u \succeq_\tau^X v$
  - Replace $\succeq_\tau^X$ by $\succeq^X$. Then, in S1, we have:
    (1) $\mathsf{f}(\mathsf{a}) \succ_\tau \mathsf{gaa}$ since $\tau(\mathsf{f}(\mathsf{a})) = o \geq \tau(\mathsf{gaa}) = o$ and, by $(\mathcal{F}_b@)$:

(a) $\mathsf{f}(\mathsf{a}) \succ \mathsf{ga}$ since, by $(\mathcal{F}_b@)$:
(i) $\mathsf{f}(\mathsf{a}) \succ \mathsf{g}$ by $(\mathcal{F}_b{>})$,
(ii) $\mathsf{f}(\mathsf{a}) \succ \mathsf{a}$ by $(\mathcal{F}_b{\triangleright})$,
(b) $\mathsf{f}(\mathsf{a}) \succ \mathsf{a}$ by 1(a)ii,
(2) $\mathsf{gaa} \succ_\tau (\lambda x\mathsf{f}(x))\mathsf{a}$ since $\tau(\mathsf{gaa}) = o \geq \tau((\lambda x\mathsf{f}(x))\mathsf{a}) = o$ and, by $(@{=})$:
(a) $\mathsf{ga} \succ_\tau \lambda x\mathsf{f}(x)$ since $\tau(\mathsf{ga}) = o \to o \geq \tau(\lambda x\mathsf{f}(x)) = o \to o$ and, by relaxing $(@{\triangleright})$:
(i) $\mathsf{a} \succ \lambda x\mathsf{f}(x)$ since, by $(\mathcal{F}_b\lambda)$:
$\mathsf{a} \succ^{\{x\}} \mathsf{f}(x)$ since, by $(\mathcal{F}_b{>})$, $\mathsf{a} \succ^{\{x\}} x$ by $(\mathcal{F}_b\mathcal{X})$,
(3) $(\lambda x\mathsf{f}(x))\mathsf{a} \succ_\tau \mathsf{f}(\mathsf{a})$ since $\tau((\lambda x\mathsf{f}(x))\mathsf{a}) = o \geq \tau(\mathsf{f}(\mathsf{a})) = o$ and by $(@\beta)$.

- $(@{=})$ $tu \succ^X t'u'$ if if $t = t'$ and $u \succ^X u'$, or $tu \succ^X_@ t'$ and $tu \succ^X_@ u'$,
where $tu \succ^X_@ v$ if $t \succ^X_\tau v$ or $u \succeq^X_\tau v$ or $tu \succ^X_\tau v$ $tu\,(\succ_\tau)_{\mathrm{mul}}\,t'u'$.
  - Replace $tu \succ^X_\tau t'$ by $tu \succ^X t'$. Then, taking $t : o \to o$, we get $tu \succ_\tau tu$ since, by relaxing $(@{=})$, we have $tu \succ t$ by $(@{\triangleright})$.
  - Replace $tu \succ^X_\tau t'$ by $tu \succ^X t'$. Then, taking $t : (o \to o) \to o$, we get $tu \succ_\tau tu$ since, by relaxing $(@{=})$, we have $tu \succ u$ by $(@{\triangleright})$.
  - We found no counter-example yet for the other cases.

- $(@\lambda)$ $tu \succ^X \lambda yv$ if $tu \succ^X v^z_y$, $\tau(y) = \tau(z)$ and z fresh
  - Replace $\succ^X$ by $\succ^{X\cup\{z\}}$. Then, in S1, we have:
(1) $\mathsf{f}(\mathsf{a}) \succ_\tau \mathsf{gaa}$ since $\tau(\mathsf{f}(\mathsf{a})) = o \geq \tau(\mathsf{gaa}) = o$ and, by $(\mathcal{F}_b@)$:
(a) $\mathsf{f}(\mathsf{a}) \succ \mathsf{ga}$ since, by $(\mathcal{F}_b@)$:
(i) $\mathsf{f}(\mathsf{a}) \succ \mathsf{g}$ by $(\mathcal{F}_b{>})$,
(ii) $\mathsf{f}(\mathsf{a}) \succ \mathsf{a}$ by $(\mathcal{F}_b{\triangleright})$,
(b) $\mathsf{f}(\mathsf{a}) \succ \mathsf{a}$ by 1(a)ii,
(2) $\mathsf{gaa} \succ_\tau (\lambda x\mathsf{f}(x))\mathsf{a}$ since $\tau(\mathsf{gaa}) = o \geq \tau((\lambda x\mathsf{f}(x))\mathsf{a}) = o$ and, by $(@{=})$:
(a) $\mathsf{ga} \succ_\tau \lambda x\mathsf{f}(x)$ since $\tau(\mathsf{ga}) = o \to o \geq \tau(\lambda x\mathsf{f}(x)) = o \to o$ and, by relaxing $(@\lambda)$:
(i) $\mathsf{ga} \succ^{\{x\}} \mathsf{f}(x)$ since, by $(@{\triangleright})$:
$\mathsf{a} \succ^{\{x\}} \mathsf{f}(x)$ since, by $(\mathcal{F}_b{>})$, $\mathsf{a} \succ^{\{x\}} x$ by $(\mathcal{F}_b\mathcal{X})$,
(3) $(\lambda x\mathsf{f}(x))\mathsf{a} \succ_\tau \mathsf{f}(\mathsf{a})$ since $\tau((\lambda x\mathsf{f}(x))\mathsf{a}) = o \geq \tau(\mathsf{f}(\mathsf{a})) = o$ and by $(@\beta)$.

- $(\lambda{\triangleright})$ $\lambda xt \succ^X v$ if $t^y_x \succeq^X_\tau v$, $\tau(x) = \tau(y)$ and y fresh
  - Replace $\succeq^X_\tau$ by $\succeq^X$. Then, in S2, we have:
(1) $\mathsf{f}(\mathsf{a}) \succ_\tau \mathsf{j}(\lambda x\lambda y\mathsf{a})$ since $\tau(\mathsf{f}(\mathsf{a})) = o \geq \tau(\mathsf{j}(\lambda x\lambda y\mathsf{a})) = o$ and, by $(\mathcal{F}_b{>})$:
(a) $\mathsf{f}(\mathsf{a}) \succ \lambda x\lambda y\mathsf{a}$ since, by $(\mathcal{F}_b\lambda)$ twice:
(i) $\mathsf{f}(\mathsf{a}) \succ^{\{x,y\}} \mathsf{a}$ by $(\mathcal{F}_b{\triangleright})$,
(2) $\mathsf{j}(\lambda x\lambda y\mathsf{a}) \succ_\tau (\lambda z\mathsf{f}(z))\mathsf{a}$ since $\tau(\mathsf{j}(\lambda x\lambda y\mathsf{a})) = o \geq \tau((\lambda z\mathsf{f}(z))\mathsf{a}) = o$ and, by $(\mathcal{F}_b@)$:
(a) $\mathsf{j}(\lambda x\lambda y\mathsf{a}) \succ \lambda z\mathsf{f}(z)$ since, by $(\mathcal{F}_b{\triangleright})$:
(i) $\lambda x\lambda y\mathsf{a} \succ_\tau \lambda z\mathsf{f}(z)$ since $\tau(\lambda x\lambda y\mathsf{a}) = o' \to o \to o \geq \tau(\lambda z\mathsf{f}(z)) = o \to o$ and, by relaxed $(\lambda{\triangleright})$, $x \notin \mathrm{FV}(\lambda z\mathsf{f}(z))$ and:
$\lambda y\mathsf{a} \succ \lambda z\mathsf{f}(z)$ since, by relaxed $(\lambda{\triangleright})$, $y \notin \mathrm{FV}(\lambda z\mathsf{f}(z))$ and $\mathsf{a} \succ \lambda z\mathsf{f}(z)$ since, by $(\mathcal{F}_b\lambda)$, $\mathsf{a} \succ^{\{z\}} \mathsf{f}(z)$ since, by $(\mathcal{F}_b{>})$, $\mathsf{a} \succ^{\{z\}} z$ by $(\mathcal{F}_b\mathcal{X})$,
(b) $\mathsf{j}(\lambda x\lambda y\mathsf{a}) \succ \mathsf{a}$ since, by $(\mathcal{F}_b{\triangleright})$:

(i) $\lambda x\lambda y\mathsf{a} \succ_\tau \mathsf{a}$ since $\tau(\lambda x\lambda y\mathsf{a}) = o' \to o \to o \geq \tau(\mathsf{a}) = o$ and, by $(\lambda\triangleright)$, $x \notin \mathrm{FV}(\mathsf{a})$ and:
$\lambda y\mathsf{a} \succ_\tau \mathsf{a}$ since $\tau(\lambda y\mathsf{a}) = o \to o \geq \tau(\mathsf{a}) = o$ and, by $(\lambda\triangleright)$again.
(3) $(\lambda z\mathsf{f}(z))\mathsf{a} \succ_\tau \mathsf{f}(\mathsf{a})$ since $\tau((\lambda z\mathsf{f}z)\mathsf{a}) = o \geq \tau(\mathsf{f}(\mathsf{a})) = o$ and, by $(@\beta)$.

- $(\lambda\neq)$ $\lambda xt \succ^X \lambda yv$ if $\lambda xt \succ^X v_y^z$, $\tau(y) \neq \tau(z)$ and z fresh
  - Replace $\succ^X$ by $\succ^{X\cup\{z\}}$. Then, in S2, we have:
    (1) $\mathsf{f}(\mathsf{a}) \succ_\tau \mathsf{j}(\lambda x\lambda y\mathsf{a})$ since $\tau(\mathsf{f}(\mathsf{a})) = o \geq \tau(\mathsf{j}(\lambda x\lambda y\mathsf{a})) = o$, by $(\mathcal{F}_b{>})$:
      (a) $\mathsf{f}(\mathsf{a}) \succ \lambda x\lambda y\mathsf{a}$ since, by $(\mathcal{F}_b\lambda)$ twice:
        (i) $\mathsf{f}(\mathsf{a}) \succ \mathsf{a}$ by $(\mathcal{F}_b\triangleright)$,
    (2) $\mathsf{j}(\lambda x\lambda y\mathsf{a}) \succ_\tau (\lambda z\mathsf{f}(z))\mathsf{a}$ since, by $(\mathcal{F}_b@)$:
      (a) $\mathsf{j}(\lambda x\lambda y\mathsf{a}) \succ \lambda z\mathsf{f}(z)$ since, by $(\mathcal{F}_b\triangleright)$:
        (i) $\lambda x\lambda y\mathsf{a} \succ_\tau \lambda z\mathsf{f}(z)$ since $\tau(\lambda x\lambda y\mathsf{a}) = o' \to o \to o \geq \tau(\lambda z\mathsf{f}(z)) = o \to o$ and, by relaxing $(\lambda\neq)$:
        $\lambda x\lambda y\mathsf{a} \succ^{\{z\}} \mathsf{f}(z)$ since, by $(\lambda\triangleright)$, $\lambda y\mathsf{a} \succ_\tau^{\{z\}} \mathsf{f}(z)$ since $\tau(\lambda y\mathsf{a}) = o \to o \geq \tau(\mathsf{f}(z)) = o$ and by $(\lambda\triangleright)$ again, $\mathsf{a} \succ_\tau^{\{z\}} \mathsf{f}(z)$ since $\tau(\mathsf{a}) = o \geq \tau(\mathsf{f}(z)) = o$ and, by $(\mathcal{F}_b{>})$, $\mathsf{a} \succ^{\{z\}} z$ by $(\mathcal{F}_b\mathcal{X})$,
      (b) $\mathsf{j}(\lambda x\lambda y\mathsf{a}) \succ \mathsf{a}$ since, by $(\mathcal{F}_b\triangleright)$:
        (i) $\lambda x\lambda y\mathsf{a} \succ_\tau \mathsf{a}$ since $\tau(\lambda x\lambda y\mathsf{a}) = o' \to o \to o \geq \tau(\mathsf{a}) = o$ and, by $(\lambda\triangleright)$:
        $\lambda y\mathsf{a} \succ_\tau \mathsf{a}$ since $\tau(\lambda y\mathsf{a}) = o \to o \geq \tau(\mathsf{a}) = o$ and, by $(\lambda\triangleright)$ again.
    (3) $(\lambda z\mathsf{f}(z))\mathsf{a} \succ_\tau \mathsf{f}(\mathsf{a})$ by $(@\beta)$.
  - Remove the condition $\tau(x) \neq \tau(y)$.
    Then, in S1, we have $\tau(\lambda x\mathsf{a}) \geq \tau(\lambda x\mathsf{b})$ and $\lambda x\mathsf{a} \succ_\tau \lambda x\mathsf{a}$ by the relaxed $(\lambda\neq)$ since $\lambda x\mathsf{a} \succ \mathsf{a}$ by $(\lambda\triangleright)$.

5.4. **Comparison with HORPO.** In [59], the last two authors define a relation on simply-typed *polymorphic* λ-terms, $>_{\mathrm{horpo}}$, and its extension $>_{\mathrm{chorpo}}$ using the notion of computability closure introduced in [15]. In this section, we explain the differences between CPO and $>_{\mathrm{horpo}}$. We will compare CPO with $>_{\mathrm{chorpo}}$ in Section 7.3.

- **Type discipline.** $>_{\mathrm{horpo}}$ and $>_{\mathrm{chorpo}}$ are relations on simply-typed *polymorphic* λ-terms, where types may contain type variables that have to be instantiated when forming function calls, while CPO is a relation on simply-typed *monomorphic* λ-terms. In the following, we will therefore compare CPO with the monomorphic versions of $>_{\mathrm{horpo}}$ and $>_{\mathrm{chorpo}}$. Extending CPO to polymorphic types along the lines of [59] is routine.
- **Relation on types.** In [59], the relation $\geq$ on types must be a quasi-ordering satisfying the following conditions[3], where $> = \geq \setminus \geq^{-1}$ is its strict part and $\simeq = \geq \cap \geq^{-1}$ its associated equivalence relation:
  (1) $>$ is well-founded;
  (2) $T \to U \simeq V$ implies $V = T' \to U'$ with $T \simeq T'$ and $U \simeq U'$;
  (3) $T \to U > V$ implies $U \geq V$ or $V = T' \to U'$ with $T \simeq T'$ and $U > U'$;
  (4) $T \geq T'$ implies $T \to U \geq T' \to U$ and $U \to T \geq U \to T'$.

  It turns out that these conditions are inconsistent: if $T > U$ then, by (4), $T \to V > U \to V$ and, by (3), $V \geq U \to V$, which is impossible by (1) [61]. However, the

[3]Condition (2) is actually stated there as an equivalence, but its converse follows from (4).

results of [59] are still true since property (4) is only used to build the simplification ordering $\succcurlyeq$[4] used for defining the interpretation of types. Instead, now, we distinguish between $>$ which must contain $\triangleright_r$ and satisfy (3)/(typ-arrow) ((2) is always satisfied when $\leq$ is an ordering instead of a quasi-ordering), and $\succcurlyeq$ which must contain $> \cup \triangleright_l$ and be well-founded. The monotony property (4) is not required anymore.

In [59], $>_{\mathrm{horpo}}$ and $>_{\mathrm{chorpo}}$ are proved well-founded not only on well-typed terms but on a larger set of terms called candidate terms, obtained by identifying equivalent types. Since, by (2), the arrow structure of equivalent types is invariant, the quotient of the set of types by $\simeq$ can be obtained by simply identifying sorts equivalent in $\simeq$, and the quasi-order becomes then an order in the quotient structure. Since rewriting on candidate terms coincides with rewriting in the quotient, an order on types suffices, which removes the need for candidate terms and their technicalities.

- **Relation on terms.** One important difference between HORPO and CPO is that, in all sub-derivations of $>_{\mathrm{horpo}}$, types must decrease ($t >_{\mathrm{horpo}} u$ only if $\tau(t) \geq \tau(u)$) while, in CPO, this is not the case: types must be checked only in case the recursive call takes a subterm of the lefthand side term (except in ($@\triangleright$) for the left argument of an application). Indeed, CPO is an optimized version of $>_{\mathrm{horpo}}$ in this respect.

  $>_{\mathrm{horpo}}$ is defined by a set of 12 rules and each rule but (9) is implied by a rule of CPO: (1) is implied by ($\mathcal{F}_b\triangleright$), (2) by ($\mathcal{F}_b>$), (3) by ($\mathcal{F}_b=$) with $\mathrm{stat}(\mathsf{f}) = \mathrm{mul}$, (4) by ($\mathcal{F}_b=$) with $\mathrm{stat}(\mathsf{f}) = \mathrm{lex}$, (5) by ($@\triangleright$), (6) by ($\lambda\triangleright$), (7) by ($\mathcal{F}_b@$), (8) by ($\mathcal{F}_b\lambda$) (HORPO requires the strong condition $x \notin \mathrm{FV}(v)$ since it does not manage bound variables; this is however done by the computability closure in CHORPO), (10) by ($\lambda=$), (11) by ($@\beta$) and (12) by ($\lambda\eta$).

  Rule (9) compares st and $u_1 \dots u_n$ with $n \geq 2$ by comparing the multisets $\{\!\{s,t\}\!\}$ and $\{\!\{u_1,\dots,u_n\}\!\}$. It is implied by ($@=$). Indeed, in this case, for all i, either $s \succeq_\tau u_i$ or $t \succeq_\tau u_i$. If there is no i such that $s = u_i$ then, for all i, $s \succ_\tau u_i$ or $t \succeq_\tau u_i$, in which case one can prove that $st \succ_\tau u_1 \dots u_k$ by induction on k. If $s = u_1$ then, for all $i \geq 2$, $t \succ_\tau u_i$, in which case one can also prove that $st \succ_\tau u_1 \dots u_k$ by induction on k. Otherwise, there is $i > 1$ such that $s = u_i$ and $t \succ_\tau u_1$. But, then, $\tau(s) \succcurlyeq \tau(t) \geq \tau(u_1) \succcurlyeq \tau(s)$, which is not possible by (typ-sn).

  On the other hand, the CPO rules ($\lambda\neq$), ($@\lambda$), ($\mathcal{F}_b\mathcal{X}$), ($@\mathcal{X}$), ($\lambda\mathcal{X}$) have no counterpart in HORPO. Therefore, HORPO is strictly included in CPO.

5.5. **Implementation.** All examples given in the paper have been checked by our implementation, which is available from the web at `http://www.lsi.upc.edu/~albert/cpo.zip`. In this implementation the precedence and the status should be provided by the user. The implemented prototype includes core CPO as well as the extended versions of the ordering defined in Section 7 and 8. Several more examples are also included together with the implementation showing the power of the developed orderings. However, like RPO, CPO cannot be compared with transformation techniques based on, for instance, the computation of dependency pairs [2, 42], but its power shows that it should be the path ordering of choice for solving the (monotonic) ordering comparisons which are generated by these transformation techniques.

[4] Written $\succeq^{\rightarrow}_{\mathcal{T}_S}$ in Lemma 3.15 of [59].

Given a precedence and a status for every function symbol, deciding if a term s is smaller than a term t in core CPO can be made in quadratic time (using a dynamic programming algorithm) if ($@\beta$) is not used. The proof is basically the same as for RPO [66]. Our prototype implementation written in Prolog does not use dynamic programming. Still, some standard optimizations over the given presentation are made, which mainly affect case ($\mathcal{F}_b{=}$). Let us split this case in two new cases, one for multiset status ($\mathcal{F}_b{=}$mul) and one for lexicographic status ($\mathcal{F}_b{=}$lex), and show that even after removing all or part of the condition $\mathsf{f}(\vec{t}) \succ^X \vec{u}$, the conjunction of both cases is equivalent to the original one.

$$\begin{array}{rl}
(\mathcal{F}_b{=}\text{mul}) & \mathsf{f}(\vec{t}) \succ^X \mathsf{g}(\vec{u}) \text{ if } \mathsf{f} \simeq_{\mathcal{F}} \mathsf{g}, \text{stat}(\mathsf{f}) = \text{mul and } \vec{t}\,(\succ_\tau)_{\text{mul}}\,\vec{u} \\
(\mathcal{F}_b{=}\text{lex}) & \mathsf{f}(\vec{t}) \succ^X \mathsf{g}(\vec{u}) \text{ if } \mathsf{f} \simeq_{\mathcal{F}} \mathsf{g}, \text{stat}(\mathsf{f}) = \text{lex and:} \\
& (\exists i)\ t_i \succ_\tau u_i \wedge (\forall j < i)\, t_j = u_j \wedge (\forall j > i)\, \mathsf{f}(\vec{t}) \succ^X u_j
\end{array}$$

In case ($\mathcal{F}_b{=}$mul), $\vec{t}\,(\succ_\tau)_{\text{mul}}\,\vec{u}$ implies that, for every u_j, there is t_i such that $t_i \succeq_\tau u_j$, which implies that $\mathsf{f}(\vec{t}) \succ^X u_j$ by ($\mathcal{F}_b{\triangleright}$). Similarly, in case ($\mathcal{F}_b{=}$lex), there is i such that $t_i \succ_\tau u_i$, $(\forall j < i)\, t_j = u_j$ and $(\forall j > i)\, \mathsf{f}(\vec{t}) \succ^X u_j$, which implies $(\forall j)\, \mathsf{f}(\vec{t}) \succ^X u_j$ by ($\mathcal{F}_b{\triangleright}$). Therefore, we have $\mathsf{f}(\vec{t}) \succ^X \vec{u}$ and hence case ($\mathcal{F}_b{=}$) can be applied as well.

As said, our implementation assumes that the precedence on function symbols and the status is given. Generating the precedence and the status automatically is a harder problem, and closely relates to the decision problem of solving ordering constraints, which is already NP-complete for RPO [76, 75], but which is nowadays efficiently done in practice by encoding the problem into SAT [26]. These kind of encodings can be easily adapted to CPO, as done for HORPO in termination tools like WANDA [63] and THOR [23].

6. Well-foundedness of core CPO

We now move to a technical analysis of the most important properties of core CPO.

6.1. Basic properties of core CPO.

Lemma 6.1. *$\succ^X$ is well-defined.*

Proof. $a \succ^X b$ is well-defined by induction on the pair (a, b) with $(=_\alpha{\triangleright} \cup \to_\beta \cup \to_\eta, =_\alpha{\triangleright})_{\text{lex}}$ as well-founded relation, where $\triangleleft$ is the subterm relation. □

Definition 6.2 (Monotony)**.** We say that $\succ_\tau$ is *monotone* if the following properties hold:

(1) if $\mathsf{f}^{|\vec{T}|} : \vec{T} \to U$, $\vec{t} : \vec{T}$, $\vec{t}' : \vec{T}$ and $\vec{t}\,(\succ_\tau)_{\text{prod}}\,\vec{t}'$, then $\mathsf{f}(\vec{t}) \succ_\tau \mathsf{f}(\vec{t}')$;
(2) if $t : U \to V$, $t' : U \to V'$, $t \succ_\tau t'$, $u : U$ and $V \geq V'$, then $tu \succ_\tau t'u$;
(3) if $t : U \to V$, $u, u' : U$ and $u \succ_\tau u'$, then $tu \succ_\tau tu'$;
(4) if $t : T$, $t' : T'$, $t \succ_\tau t'$ and $\tau(x) \to T \geq \tau(x) \to T'$, then $\lambda x t \succ_\tau \lambda x t'$.

Lemma 6.3. *$\succ_\tau$ is monotone.*

Proof.

(1) Since $\tau(\mathsf{f}(\vec{t})) = \tau(\mathsf{f}(\vec{t}'))$, it suffices to check that $\mathsf{f}(\vec{t}) \succ \mathsf{f}(\vec{t}')$. By Lemma 4.1, $\vec{t}\,(\succ_\tau)_{\text{stat}(\mathsf{f})}\,\vec{t}'$. By ($\mathcal{F}_b{\triangleright}$), $\mathsf{f}(\vec{t}) \succ \vec{t}'$ since, for each i, $t_i \succeq_\tau t'_i$. We conclude by ($\mathcal{F}_b{=}$).
(2) Since $\tau(tu) \geq \tau(t'u)$, it suffices to check that $tu \succ t'u$. This follows by by ($@{=}$).
(3) Since $\tau(tu) = \tau(tu')$, it suffices to check that $tu \succ tu'$. This follows by ($@{=}$).
(4) By ($\lambda{=}$). □

Note that Lemma 6.3 holds for any relation satisfying $(\mathcal{F}_b\triangleright)$, $(\mathcal{F}_b=)$, $(@=)$ and $(\lambda=)$.

Lemma 6.4. *If $a \succ^X b$, then $\mathrm{FV}(b) \subseteq \mathrm{FV}(a) \cup X$.*

Proof. By an easy induction on $a \succ^X b$. We detail a selection of cases:

- $(\mathcal{F}_b\lambda)$ By the induction hypothesis, $\mathrm{FV}(v_y^z) \subseteq \mathrm{FV}(\mathsf{f}(\vec{t})) \cup X \cup \{y\}$. Now, $\mathrm{FV}(\lambda yv) = \mathrm{FV}(v_y^z) \setminus \{z\}$ since z is fresh. The result follows.
- $(\lambda\triangleright)$ By the induction hypothesis, $\mathrm{FV}(v) \subseteq \mathrm{FV}(t_x^z) \cup X$. Therefore, $\mathrm{FV}(v) \subseteq \mathrm{FV}(\lambda xt) \cup X$ since $\mathrm{FV}(t_x^z) \subseteq \mathrm{FV}(\lambda xt) \cup \{z\}$ and $z \notin \mathrm{FV}(v)$.
- $(\lambda=)$ By the induction hypothesis, $\mathrm{FV}(v_y^z) \subseteq \mathrm{FV}(t_x^z) \cup X$. Now, $\mathrm{FV}(t_x^z) \subseteq \mathrm{FV}(\lambda xt) \cup \{z\}$ and, either $y \in \mathrm{FV}(v)$ and $\mathrm{FV}(v_y^z) = \mathrm{FV}(\lambda yv) \cup \{z\}$, or $\mathrm{FV}(v_y^z) = \mathrm{FV}(\lambda yv)$. Therefore, $\mathrm{FV}(\lambda yv) \subseteq \mathrm{FV}(\lambda xt) \cup X$ since $z \notin \mathrm{FV}(\lambda yv)$.
- $(\lambda\neq)$ By the induction hypothesis, $\mathrm{FV}(v_y^z) \subseteq \mathrm{FV}(\lambda xt) \cup X$. Since z is fresh for λxt, X and λyv, $y \notin \mathrm{FV}(v)$. Therefore, $\mathrm{FV}(\lambda yv) = \mathrm{FV}(v) - \{y\} \subseteq \mathrm{FV}(\lambda xt) \cup X$. □

Lemma 6.5. *If $a \succ^X b$, $a =_\alpha a'$ and $b =_\alpha b'$, then $a' \succ^X b'$.*

Proof. We prove (i) $a \succ^X b$ and $a =_\alpha a'$ implies $a' \succ^X b$, and (ii) $a \succ^X b$ and $b =_\alpha b'$ implies $a \succ^X b'$, separately by induction on $a \succ^X b$. We only detail some cases:

- $(\mathcal{F}_b\lambda)$ (ii) Assume that $\lambda yv =_\alpha b'$. Then, there are y' and v' such that $b' = \lambda y'v'$, $y \notin \mathrm{FV}(\lambda y'v')$ and $v =_\alpha {v'}_{y'}^{y}$. Hence, $v_y^z =_\alpha {v'}_{y'y}^{y\,z} =_\alpha {v'}_{y'}^{z}$ and, by the induction hypothesis, $\mathsf{f}(\vec{t}) \succ^{X\cup\{z\}} {v'}_{y'}^{z}$. Now, $z \notin \mathrm{FV}(\lambda y'v') \cup \mathrm{FV}(\mathsf{f}(\vec{t}))$ since $\mathrm{FV}(\lambda y'v') = \mathrm{FV}(\lambda yv)$ and $z \notin \mathrm{FV}(\lambda yv) \cup \mathrm{FV}(\mathsf{f}(\vec{t}))$. Therefore, by $(\mathcal{F}_b\lambda)$, $\mathsf{f}(\vec{t}) \succ^X \lambda y'v'$.
- $(\lambda\triangleright)$ (ii) Assume that $v =_\alpha v'$. By the induction hypothesis, $t_x^z \succeq_\tau^X v'$. Now, z is fresh for λxt, X and v' since $\mathrm{FV}(v') = \mathrm{FV}(v)$ and z is fresh for λxt, X and v. Therefore, by $(\lambda\triangleright)$, $\lambda xt \succ^X v'$.

  (i) Assume now that $\lambda xt =_\alpha a'$. Then, there are x' and t' such that $a' = \lambda x't'$, $x \notin \mathrm{FV}(\lambda x't')$ and $t =_\alpha {t'}_{x'}^{x}$. Hence, $t_x^z =_\alpha {t'}_{x'x}^{x\,z} =_\alpha {t'}_{x'}^{z}$ and, by the induction hypothesis, ${t'}_{x'}^{z} \succeq_\tau^X v$. Now, z is fresh for $\lambda x't'$, X and v since $\mathrm{FV}(\lambda x't') = \mathrm{FV}(\lambda xt)$ and z is fresh for λxt, X and v. Therefore, by $(\lambda\triangleright)$, $\lambda x't' \succ^X v$.
- $(\lambda=)$ (ii) Assume that $\lambda yv =_\alpha b'$. Then, there are y' and v' such that $b' = \lambda y'v'$, $y \notin \mathrm{FV}(\lambda y'v')$ and $v =_\alpha {v'}_{y'}^{y}$. Hence, $v_y^z =_\alpha {v'}_{y'y}^{z\,z} =_\alpha {v'}_{y'}^{z}$ and, by the induction hypothesis, $t_x^z \succ^X {v'}_{y'}^{z}$. Now, z is fresh for λxt, X and $\lambda y'v'$ since $\mathrm{FV}(\lambda y'v') = \mathrm{FV}(\lambda yv)$ and z is fresh for λxt, X and λyv. Therefore, by $(\lambda=)$, $\lambda xt \succ^X \lambda y'v'$.
- $(\lambda\neq)$ (ii) Assume that $\lambda yv =_\alpha b'$. Then, there are y' and v' such that $b' = \lambda y'v'$, $y' \notin \mathrm{FV}(\lambda yv)$ and ${v}_{y}^{y'} =_\alpha v'$. By Lemma 6.4, $y \notin \mathrm{FV}(v)$. Hence, $y' \notin \mathrm{FV}(v')$, $v_y^z =_\alpha {v'}_{y'}^{z}$ and, by the induction hypothesis, $\lambda xt \succ^X {v'}_{y'}^{z}$. Moreover, z is fresh for λxt, X and $\lambda y'v'$. Therefore, by $(\lambda\neq)$, $\lambda xt \succ^X \lambda y'v'$. □

Hence, if $t \succ^X u$ and V is a finite set of variables, then one can always assume without lost of generality that the bounding variables of t and u do not belong to V.

Invariance by variable renaming can also be extended to X:

Lemma 6.6. *Assume that $t \succ^X u$.*

(1) *If σ is away from X, then $t\sigma \succ^X u\sigma$.*
(2) *If $e \in X$, $e' \notin \mathrm{FV}(\lambda eu)$ and $\tau(e) = \tau(e')$, then $t \succ^{X-\{e\}\cup\{e'\}} u_e^{e'}$.*

Proof. Note that substitution preserves typing ($\tau(t\sigma) = \tau(t)$). Let $X_e^{e'} = X - \{e\} \cup \{e'\}$. Wlog we can assume that $e \neq e'$. Hence, $e' \notin \mathrm{FV}(u)$. We now proceed by induction on the deduction height of $t \succ^X u$. We only detail some cases:

(1) ($\mathcal{F}_b$=) By induction hypothesis and Lemma 4.1, $(\forall i)\ \mathsf{f}(\vec{t})\sigma \succ^X u_i\sigma$ and $\vec{t}\sigma\,(\succ_\tau)_{\mathrm{stat}(\mathsf{f})}\,\vec{u}\sigma$. Therefore, by ($\mathcal{F}_b$=), $\mathsf{f}(\vec{t})\sigma \succ^X \mathsf{g}(\vec{u})\sigma$.

($\mathcal{F}_b\lambda$) Wlog we can assume σ away from $\{y\}$. Hence, $(\lambda yv)\sigma = \lambda y(v\sigma)$. Let now z' be a variable of the same type as z, fresh for $\mathsf{f}(\vec{t})\sigma$, X, $\lambda y(v\sigma)$ and $\lambda z v_y^z =_\alpha \lambda yv$, and such that σ is away from $\{z'\}$. By induction hypothesis (2), $\mathsf{f}(\vec{t}) \succ^{(X\cup\{z\})_z^{z'}} (v_y^z)_z^{z'}$. Since $z \notin X$, $(X \cup \{z\})_z^{z'} = X \cup \{z'\}$. Since $\binom{z'}{z}$ is away from $\{y\}$ and $z \notin \mathrm{FV}(v)$, $(v_y^z)_z^{z'} = v_y^{z'}$. By induction hypothesis, $\mathsf{f}(\vec{t})\sigma \succ^{X\cup\{z'\}} (v_y^{z'})\sigma$. Since σ is away from $\{y, z'\}$, $(v_y^{z'})\sigma = (v\sigma)_y^{z'}$. Therefore, by ($\mathcal{F}_b\lambda$), $\mathsf{f}(\vec{t})\sigma \succ^X \lambda y(v\sigma)$.

($\lambda\triangleright$) Wlog we can assume that σ is away from $\{x\}$. Hence, $(\lambda xt)\sigma = \lambda x(t\sigma)$. After Lemma 6.4 and since σ is away from X, $\mathrm{FV}(v) \subseteq \mathrm{FV}(\lambda xt)$. Hence, we can also assume wlog that $\mathrm{dom}(\sigma) \subseteq \mathrm{FV}(\lambda xt)$. Let now z' be a variable of the same type as z, fresh for $\lambda x(t\sigma)$, X, $v\sigma$ and x, and such that σ is away from $\{z'\}$. Let $\sigma' = \sigma \cup \{(z, z')\}$. Since σ' is away from X, by induction hypothesis, $(t_x^z)\sigma' \succ^X v\sigma'$. Since $\mathrm{dom}(\sigma) \subseteq \mathrm{FV}(\lambda xt)$ and σ is away from $\{x, z'\}$, $(t_x^z)\sigma' = (t\sigma)_x^{z'}$, and since $z \notin \mathrm{FV}(v)$, $v\sigma' = v\sigma$. Therefore, by (λ=), $\lambda x(t\sigma) \succ^X v\sigma$.

(λ=) Wlog we can assume that σ is away from $\{x, y\}$. Hence, $(\lambda xt)\sigma = \lambda x(t\sigma)$ and $(\lambda yv)\sigma = \lambda y(v\sigma)$. After Lemma 6.4 and since σ is away from X, $\mathrm{FV}(\lambda yv) \subseteq \mathrm{FV}(\lambda xt)$. Hence, we can also assume wlog that $\mathrm{dom}(\sigma) \subseteq \mathrm{FV}(\lambda xt)$. Let now z' be a variable of the same type as z, fresh for $\lambda x(t\sigma)$, X, $\lambda y(v\sigma)$, x and y, and such that σ is away from $\{z'\}$. Let $\sigma' = \sigma \cup \{(z, z')\}$. Since σ' is away from X, by induction hypothesis, $(t_x^z)\sigma' \succ^X (v_y^z)\sigma'$. Since $\mathrm{dom}(s) \subseteq \mathrm{FV}(\lambda xt)$, $\mathrm{dom}(\sigma) \subseteq \mathrm{FV}(\lambda yv)$, and σ is away from $\{x, y, z'\}$, $(t_x^z)\sigma' = (t\sigma)_x^{z'}$ and $(v_y^z)\sigma' = (v\sigma)_y^{z'}$. Therefore, by ($\lambda$=), $(\lambda xt)\sigma \succ^X (\lambda yv)\sigma$.

($\lambda\neq$) Wlog we can assume σ away from $\{x, y\}$. Hence, $(\lambda xt)\sigma = \lambda x(t\sigma)$ and $(\lambda yv)\sigma = \lambda y(v\sigma)$. Let z' fresh for $\lambda xt\sigma$, X and $\lambda yv\sigma$. By Lemma 6.4, $y \notin \mathrm{FV}(v)$, hence $v_y^z = v_y^{z'}$. By induction hypothesis, $(\lambda xt)\sigma \succ^X v_y^{z'}\sigma$. Therefore, by ($\lambda\neq$), $\lambda x(t\sigma) \succ^X \lambda y(v\sigma)$.

(2) ($\mathcal{F}_b\lambda$) Wlog we can assume $\binom{e'}{e}$ away from $\{y\}$. Hence, $(\lambda yv)_e^{e'} = \lambda y(v_e^{e'})$. Let now z' be a variable of the same type as z, fresh for $\mathsf{f}(\vec{t})$, $X_e^{e'}$, $\lambda y(v_e^{e'})$, $\lambda z v_y^z =_\alpha \lambda yv$ and y. By induction hypothesis, $\mathsf{f}(\vec{t}) \succ^{(X\cup\{z\})_z^{z'}} (v_y^z)_z^{z'}$. Since $z \notin X$, $(X\cup\{z\})_z^{z'} = X\cup\{z'\}$. Since $z \notin \mathrm{FV}(\lambda yv)$, $(v_y^z)_z^{z'} = v_y^{z'}$. By induction hypothesis, $\mathsf{f}(\vec{t}) \succ^{(X\cup\{z'\})_e^{e'}} (v_y^{z'})_e^{e'}$. Since $\binom{e'}{e}$ is away from $\{y, z'\}$, $(X \cup \{z'\})_e^{e'} = X_e^{e'} \cup \{z'\}$ and $(v_y^{z'})_e^{e'} = (v_e^{e'})_y^{z'}$. Therefore, by ($\mathcal{F}_b\lambda$), $\mathsf{f}(\vec{t}) \succ^{X_e^{e'}} \lambda y v_e^{e'}$.

($\lambda\triangleright$) Let z' be a variable of the same type as z, fresh for λxt, X, $X_e^{e'}$, $v_e^{e'}$ and x. Since $\binom{z'}{z}$ is away from X, by induction hypothesis (1), $(t_x^z)_z^{z'} \succeq_\tau^X v_z^{z'}$. Since $z' \neq x$ and $z \notin \mathrm{FV}(t)$, $(t_x^z)_z^{z'} = t_x^{z'}$. Since $z \notin \mathrm{FV}(v)$, $v_z^{z'} = v$. So, by induction hypothesis, $t_x^{z'} \succeq_\tau^{X_e^{e'}} v_e^{e'}$. Therefore, by ($\lambda\triangleright$), $\lambda xt \succ^{X_e^{e'}} v_e^{e'}$.

(λ=) Wlog we can assume $\binom{e'}{e}$ away from $\{y\}$. Hence, $(\lambda yv)_e^{e'} = \lambda y(v_e^{e'})$. Let now z' be a variable of the same type as z, fresh for λxt, X, $X_e^{e'}$, $\lambda y v_e^{e'}$, x and y. Since $\binom{z'}{z}$ is away

from X, by induction hypothesis (1), $(t_x^z)_z^{z'} \succ^X (v_y^z)_z^{z'}$. Since $z' \neq x$ and $z \notin \mathrm{FV}(t)$, $(t_x^z)_z^{z'} = t_x^{z'}$. Since $z' \neq y$ and $z \notin \mathrm{FV}(v)$, $(v_y^z)_z^{z'} = v_y^{z'}$. So, by induction hypothesis, $t_x^{z'} \succ^{X_e^{e'}} (v_y^{z'})_e^{e'}$. Since (e') is away from $\{y, z'\}$, $(v_y^{z'})_e^{e'} = (v_e^{e'})_y^{z'}$. Therefore, by $(\lambda{=})$, $\lambda x t \succ^{X_e^{e'}} \lambda y(v_e^{e'})$.

$(\lambda{\neq})$ Wlog we can assume (e') away from $\{y\}$. Hence, $(\lambda y v)_e^{e'} = \lambda y(v_e^{e'})$. Let z' fresh for $\lambda x t$, $X_e^{e'}$ and $\lambda y(v_e^{e'})$. By Lemma 6.4, $y \notin \mathrm{FV}(v)$, hence $v_y^z = v_y^{z'}$. By the induction hypothesis, $\lambda x t \succ^{X_e^{e'}} (v_y^{z'})_e^{e'} = (v_e^{e'})_y^{z'}$. Therefore, by $(\lambda{\neq})$, $\lambda x t \succ^{X_e^{e'}} \lambda y(v_e^{e'})$. □

6.2. **Tait and Girard's computability.** We now turn to the proof that $\succ_\tau$ is well-founded. This proof is based on the meticulous analysis of the technique of computability predicates of Tait and Girard for proving the termination of β-reduction in typed λ-calculi [83, 45, 84, 46]. This technique consists in the following three steps:

(1) interpret each type T by a set $[\![T]\!]$ of so-called computable terms;
(2) prove that, for every type T, $[\![T]\!]$ satisfies some properties among which termination, *i.e.* $[\![T]\!] \subseteq \mathrm{SN}(\succ_\tau)$;
(3) prove that every (well typed) term is computable.

For arrow types, we will use the standard interpretation but, for sorts, we *a priori* have some freedom and we will indeed use this freedom to extend CPO to inductive types later in Section 7.

Definition 6.7 (Computability)**.** A base type interpretation is a map $I : \mathcal{S} \to \mathcal{P}(\mathcal{L})$ such that, for all sorts A, $I(\mathsf{A})$ is a set of terms of type A. A base type interpretation naturally extends to types as follows:

- $[\![\mathsf{A}]\!]_I = I(\mathsf{A})$
- $[\![U \to V]\!]_I = \{t \in \mathcal{L} \mid t : U \to V \wedge (\forall u) u \in [\![U]\!]_I \Rightarrow tu \in [\![V]\!]_I\}$

Given a base type interpretation I, we say that:

- a term $t : T$ is *I-computable* if $t \in [\![T]\!]_I$;
- a substitution σ is I-computable on a set X of variables if, for all $x \in X$, $x\sigma$ is I-computable; it is I-computable if it is I-computable on $\mathcal{X}$;
- a function symbol $\mathsf{f}^{|\vec{T}|} : \vec{T} \Rightarrow U$ is I-computable if, for all I-computable terms $\vec{t} : \vec{T}$, $\mathsf{f}(\vec{t})$ is I-computable.

Let Σ_I be the set of pairs $(\mathsf{f}, \vec{t})$ such that $\mathsf{f} \in \mathcal{F}$, $\mathsf{f}(\vec{t})$ is a term and $\vec{t}$ are I-computable. Given a relation on terms R, let $(>_\mathcal{F}, R_{\mathrm{stat}})_{\mathrm{lex}}$ be the relation on Σ_I such that $(\mathsf{f}, \vec{t})\,(>_\mathcal{F}, R_{\mathrm{stat}})_{\mathrm{lex}}\,(\mathsf{g}, \vec{u})$ if either $\mathsf{f} >_\mathcal{F} \mathsf{g}$, or $\mathsf{f} \simeq_\mathcal{F} \mathsf{g}$ and $\vec{t}\, R_{\mathrm{stat}(\mathsf{f})}\, \vec{u}$.

Our first lemma has a straightforward proof:

Lemma 6.8. *Let I_1 and I_2 be two base type interpretations, and T be a type. Then, $[\![T]\!]_{I_1} = [\![T]\!]_{I_2}$ if I_1 and I_2 agree on every sort occurring in T.*

We then consider the following properties:

Definition 6.9 (Sets of neutral terms)**.** A set $\mathcal{N}$ is a set of *neutral* terms if it satisfies the following properties:

- $\mathcal{X} \subseteq \mathcal{N}$ **(neutral-var)**
- for all x, t, u, $(\lambda x t) u \in \mathcal{N}$ **(neutral-beta)**

- if $t \in \mathcal{N}$ then, for all u, $tu \in \mathcal{N}$ **(neutral-app)**
- for all x and t, $\lambda xt \notin \mathcal{N}$ **(neutral-not-lam)**

Definition 6.10 (Computability properties)**.** Given a base type interpretation I and a set $\mathcal{N}$ of neutral terms, a set S of terms of type T is an *I-computability predicate* if it satisfies the following properties:

- $S \subseteq \mathrm{SN}(\succ_\tau)$ **(comp-sn)**
- If $t \in S$, then every $\succ_\tau$-reduct of t is I-computable **(comp-red)**
- $t \in S$ if $t : T$, $t \in \mathcal{N}$ and every $\succ_\tau$-reduct of t is I-computable **(comp-neutral)**
- $\lambda xt \in S$ if $T = U \to V$, $\lambda xt : T$ and, for all $u \in [\![U]\!]_I$, t^u_x is I-computable **(comp-lam)**

We can then prove that every term is strongly normalizing if the sets $[\![T]\!]_I$ satisfy some of these properties and function symbols are I-computable, whatever the base type interpretation I and the set $\mathcal{N}$ of neutral terms are:

Theorem 6.11. *Given a base type interpretation I and a set $\mathcal{N}$ of neutral terms, $\succ_\tau$ is well-founded if:*

- *for every type T, $[\![T]\!]_I$ satisfies* (comp-sn)*,* (comp-neutral) *and* (comp-lam)*;*
- *every function symbol $\mathsf{f} \in \mathcal{F}$ is I-computable.*

Proof. Because, for every T, $[\![T]\!]_I$ satisfies *(comp-sn)*, it suffices to prove that every term is I-computable. By (neutral-var), variables belongs to $\mathcal{N}$. Because, for every T, $[\![T]\!]_I$ satisfies *(comp-neutral)*, variables are computable. Hence, the identity substitution is computable. We then prove that, for all $t : T$ and computable σ, $t\sigma \in [\![T]\!]_I$, by induction on t.

- $t = x$. Then, $t\sigma = x\sigma$ is computable since σ is computable.
- $t = uv$. By induction hypothesis, $u\sigma$ and $v\sigma$ are computable. Therefore, $t\sigma = (u\sigma)(v\sigma)$ is computable.
- $t = \lambda xu$ with $x : V$. By renaming x, we can assume that σ is away from $\{x\}$. Hence, $t\sigma = \lambda x(u\sigma)$. By assumption, $[\![T]\!]_I$ satisfies (comp-lam). Therefore, $t\sigma$ is computable if, for all computable $v : V$, $(u\sigma)^v_x$ is computable. Since σ is away from $\{x\}$, $(u\sigma)^v_x = u(\sigma \cup \{(x, v)\})$ which is computable by induction hypothesis.
- $t = \mathsf{f}(\vec{t})$ with $\mathsf{f}^{|\vec{T}|} : \vec{T} \to U$. By induction hypothesis, $\vec{t}\sigma$ are computable. Thus, $(\mathsf{f}, \vec{t}\sigma) \in \Sigma$ and, by assumption, $\mathsf{f}(\vec{t})\sigma$ is computable. □

We are therefore left to find a set of neutral terms $\mathcal{N}$ and a base type interpretation I so that type interpretations are computability predicates and function symbols are computable.

First, we are going to study under which conditions the interpretation of an arrow type $U \to V$, $[\![U \to V]\!]_I$, satisfies the above computability properties, whatever the base type interpretation I and the set of neutral terms $\mathcal{N}$ are.

Second, we will define a set of neutral terms $\mathcal{N}$ and a base type interpretation I so that, for every type T, $[\![T]\!]_I$ satisfies all the computability properties. Finally, we will prove that function symbols are computable by induction on $(\succ_\mathcal{F}, (\succ_\tau)_{\mathrm{stat}})_{\mathrm{lex}}$, which is well-founded when the following conditions are satisfied:

Lemma 6.12. *Given a base type interpretation I and a well-founded relation on I-computable terms R, $(\succ_\mathcal{F}, R_{\mathrm{stat}})_{\mathrm{lex}}$ is well-founded if, for all $\mathsf{f}^{|\vec{T}|} : \vec{T} \to U$, $[\![\vec{T}]\!]_I \subseteq \mathrm{SN}(R)$.*

Proof. If $(\mathsf{f}, \vec{t}) \in \Sigma_I$, then $\vec{t} \in [\![\vec{T}]\!]_I$. By assumption, $[\![\vec{T}]\!]_I \subseteq \mathrm{SN}(R)$. Hence, by Lemma 4.1, $\vec{t} \in \mathrm{SN}(R_{\mathrm{stat}(\mathsf{g})})$ whatever g is. Assume now that there is an infinite $(\succ_\mathcal{F}, R_{\mathrm{stat}})_{\mathrm{lex}}$-decreasing

sequence $(\mathsf{f}_i, \vec{t}_i)_{i\geq 0}$. Then, $(\mathsf{f}_i)_{i\geq 0}$ is an infinite $\geq_{\mathcal{F}}$-decreasing sequence. Since $>_{\mathcal{F}}$ is well-founded by assumption, there must be some j such that, for all $i \geq j$, $\mathsf{f}_i \simeq_{\mathcal{F}} \mathsf{f}_j$. Since symbols equivalent in $\simeq_{\mathcal{F}}$ have the same status by assumption, $(\vec{t}_i)_{i\geq j}$ is an infinite $R_{\mathrm{stat}(\mathsf{f}_j)}$-decreasing sequence, a contradiction. □

6.3. **Computability properties of arrow types.** In this sub-section, the results hold for any base type interpretation I and any set of neutral terms $\mathcal{N}$. For the sake of simplicity, we drop the index I in $[\![T]\!]_I$ and simply write $[\![T]\!]$.

Lemma 6.13. $[\![U \to V]\!]$ *satisfies* (comp-sn) *if:*

- $[\![U]\!] \neq \varnothing$*, which is the case if* $[\![U]\!]$ *satisfies* (comp-neutral)*;*
- $[\![V]\!]$ *satisfies* (comp-sn)*.*

Proof. Assume that there is an infinite reduction sequence $t_0 >_\tau t_1 >_\tau \dots$ with $t_0 \in [\![U \to V]\!]$ and $t_i : T_i$. By definition of $[\![U \to V]\!]$, $T_0 = U \to V$. By definition of $>_\tau$, $T_0 \geq T_1 \geq \dots$ By assumption, $[\![U]\!] \neq \varnothing$. So, let $u \in [\![U]\!]$. By definition of $[\![U \to V]\!]$, we have $t_0 u \in [\![V]\!]$. We now prove that there is an infinite reduction sequence starting from $t_0 u$. Since $[\![V]\!]$ is assumed to satisfy (comp-sn), this is not possible. Therefore, $[\![U \to V]\!]$ satisfies (comp-sn) too. By (typ-arrow) there are only two cases:

- For all i, $T_i = U \to B_i$ for some B_i. By monotony (Lemma 6.3), $t_0 u >_\tau t_1 u >_\tau \dots$
- There is i such that T_{i+1} is a sort or $T_{i+1} = A_{i+1} \to B_{i+1}$ with $A_{i+1} \neq U$. Let k be the smallest i satisfying this condition. Hence, for all $i \leq k$, there is B_i such that $T_i = U \to B_i$. By monotony (Lemma 6.3), $t_0 u >_\tau \dots t_k u$. By (typ-arrow), we have $B_k \geq T_{k+1}$. Hence, by ($@_\triangleright$), $t_k u >_\tau t_{k+1}$. □

Lemma 6.14. $[\![U \to V]\!]$ *satisfies* (comp-red) *if:*

- $[\![U]\!] \neq \varnothing$*, which is the case if* $[\![U]\!]$ *satisfies* (comp-neutral)*;*
- $[\![V]\!]$ *satisfies* (comp-red)*.*

Proof. Let $t \in [\![U \to V]\!]$ and $t' : T'$ such that $t >_\tau t'$. By definition of $[\![U \to V]\!]$, $t : U \to V$. By definition of $>_\tau$, $U \to V \geq T'$. By (typ-arrow), there are two cases:

(1) $V \geq T'$. By assumption, $[\![U]\!] \neq \varnothing$. So, let $u \in [\![U]\!]$. By definition of $[\![U \to V]\!]$, $tu \in [\![V]\!]$. By ($@_\triangleright$), we have $tu >_\tau t'$. Therefore $t' \in [\![T']\!]$ since $[\![V]\!]$ satisfies (comp-red).
(2) $T' = U \to V'$ and $V \geq V'$. Let $u \in [\![U]\!]$. By monotony (Lemma 6.3), $tu >_\tau t'u$. By definition of $[\![U \to V]\!]$, $tu \in [\![V]\!]$. Since $[\![V]\!]$ satisfies (comp-red), $t'u \in [\![V']\!]$. Therefore, $t' \in [\![T']\!]$. □

Lemma 6.15. *Let* $t : U \to V$ *and* $u : U$*. Then, every* $>_\tau$*-reduct of* tu *is computable if:*

- *every* $>_\tau$*-reduct of* t *is computable;*
- u *is computable;*
- *if* $t = \lambda x v$*, then* v_x^u *is computable;*
- *for all* u' *such that* $u >_\tau u'$*,* tu' *is computable;*
- $[\![U]\!]$ *satisfies* (comp-red)*;*
- $[\![V]\!]$ *satisfies* (comp-red)*;*
- $[\![V']\!]$ *satisfies* (comp-lam) *whenever* $V' \leq V$*.*

Proof. We prove that every $w : W$ such that $tu >_\tau w$ is computable, by induction on $|w|$. By definition of $>_\tau$, we have $V \geq W$.

- ($@_\triangleright$)

  - $t \geq w$. By (typ-right-subterm), $U \to V > V$. Hence, by transitivity, $U \to V > W$ and $t \succ_\tau w$. Therefore, w is computable by assumption.
  - $u \succeq_\tau w$. Then, w is computable since, by assumption, u is computable and $[\![U]\!]$ satisfies (comp-red).
- (@=) $w = t'u'$ and either:
  - $t = t'$ and $u > u'$, in which case $t'u$ is computable by assumption since $u \succ_\tau u'$;
  - or $tu \succ_@ t'$ and $tu \succ_@ u'$. We prove that, for $v \in \{t', u'\}$, if $tu \succ_@ v$ then v is computable. There are three cases:
    * $t \succ_\tau v$. Then, v is computable by assumption.
    * $u \succeq_\tau v$. Then, either $u = v$ and v is computable by assumption, or $u \succ_\tau v$ and v is computable since u is computable and $[\![U]\!]$ satisfies (comp-red).
    * $tu \succ_\tau v$. Then, since $v \in \{t', u'\}$, v is computable by induction hypothesis.
- (@λ) $w = \lambda yv$, $tu > v$ and $y \notin \mathrm{FV}(v)$ by Lemma 6.4. Then, there are A and B such that $W = A \to B$. By (typ-right-subterm), $W > B$. Hence, by transitivity, $V > B$ and $tu \succ_\tau v$. Thus, by induction hypothesis, v is computable. Since $[\![W]\!]$ satisfies (comp-lam), w is computable if, for all computable term $a : A$, v_y^a is computable. Since $y \notin \mathrm{FV}(v)$, $v_y^a = v$. Therefore, w is computable.
- (@$\mathcal{X}$) $w \in \varnothing$. Impossible.
- (@β) $t = \lambda xv$ and $v_x^u \geq^\varnothing w$. Since $\tau((\lambda xv)u) = \tau(v_x^u)$, we have $v_x^u \succeq_\tau w$. Thus w is computable since v_x^u is computable and $[\![V]\!]$ satisfies (comp-red). □

Lemma 6.16. *Let $t : U \to V$ and $u : U$. Then, tu is computable if:*

- *every $\succ_\tau$-reduct of t is computable;*
- *u is computable;*
- *if $t = \lambda xv$, then v_x^u is computable;*
- *either t is neutral or $t = \lambda xv$;*
- *$[\![U]\!]$ satisfies* (comp-red) *and* (comp-sn)*;*
- *$[\![V]\!]$ satisfies* (comp-red) *and* (comp-neutral)*;*
- *$[\![V']\!]$ satisfies* (comp-lam) *whenever $V' \leq V$.*

Proof. We prove that tu is computable by induction on u with $\succ_\tau$ as well-founded relation ($[\![U]\!]$ satisfies (comp-sn) by assumption). So, by induction hypothesis, for all u' such that $u \succ_\tau u'$, tu' is computable. Hence, by Lemma 6.15, every $\succ_\tau$-reduct of tu is computable. Now, tu is neutral because, either t is neutral and tu is neutral by (neutral-app), or $t = \lambda xv$ and tu is neutral by (neutral-beta). Therefore, tu is computable since $[\![V]\!]$ satisfies (comp-neutral). □

Corollary 6.17. *$[\![U \to V]\!]$ satisfies* (comp-neutral) *if:*

- *$[\![U]\!]$ satisfies* (comp-red) *and* (comp-sn)*;*
- *$[\![V]\!]$ satisfies* (comp-red) *and* (comp-neutral)*;*
- *$[\![V']\!]$ satisfies* (comp-lam) *whenever $V' \leq V$.*

Proof. Let t be a neutral term of type $U \to V$ such that every $\succ_\tau$-reduct of t is computable. By definition, $t \in [\![U \to V]\!]$ if, for all computable $u : U$, tu is computable. Since t is neutral, by (neutral-not-lam), t is not of the form λxv. Therefore, by Lemma 6.16, tu is computable since all the required properties are satisfied. □

Lemma 6.18. *Let $x : U$ and $v : V$. Then, λxv is computable if:*

- *for all computable $u : U$, v_x^u is computable;*

- $[\![U]\!]$ *satisfies* (comp-sn) *and* (comp-red) *and contains a variable, which is the case if it satisfies* (comp-neutral) *too;*
- $[\![V]\!]$ *satisfies* (comp-sn), (comp-red) *and* (comp-neutral)*;*
- $[\![V']\!]$ *satisfies* (comp-lam) *whenever* $V' \leq V$.

Proof. By definition, λxv is computable if, for all computable $u : U$, $(\lambda xv)u$ is computable. By Lemma 6.16, $(\lambda xv)u$ is computable if every $\succ_\tau$-reduct of λxv is computable, the other conditions being satisfied. Since $[\![U]\!]$ contains a variable, we can wlog assume that this is x. So, $v^x_x = v$ is computable. Since $[\![V]\!]$ satisfies (comp-sn), $v \in \mathrm{SN}(\succ_\tau)$. We now prove that every $\succ_\tau$-reduct $w : W$ of λxv is computable, by induction on $(v, |w|)$ with $(\succ_\tau, >_{\mathbb{N}})_{\mathrm{lex}}$ as well-founded relation. By definition of $\succ_\tau$, we have $U \to V \geq W$.

- $(\lambda\triangleright)$ $v \succeq_\tau w$. Since v is computable and $[\![V]\!]$ satisfies (comp-red), we have w computable.
- $(\lambda=)$ $w = \lambda xb$ and $v \succ b$. Then, there is B such that $W = U \to B$. Since $U \to V \geq W$, by Lemma 2.4, we have $V \geq B$. Hence, $v \succ_\tau b$. Thus, by induction hypothesis, every $\succ_\tau$-reduct of λxb is computable. By Lemma 6.16, to prove that λxb is computable, it suffices to check that, for all $u : U$ computable, b^u_x is computable. By assumption, v^u_x is computable. By stability by substitution (Lemma 1), $v^u_x \succ_\tau b^u_x$. Therefore, b^u_x is computable since $[\![V]\!]$ satisfies (comp-red) by assumption.
- $(\lambda\neq)$ $w = \lambda yb$, $\tau(x) \neq \tau(y)$, $\lambda xv \succ b$ and $y \notin \mathrm{FV}(b)$ by Lemma 6.4. Then, there are A and B such that $W = A \to B$. Since $U \neq A$, by (typ-arrow), $V \geq W$. Since, by assumption, $[\![W]\!]$ satisfies (comp-lam), it suffices to prove that, for all computable $a : A$, b^a_y is computable. Since $y \notin \mathrm{FV}(b)$, $b^a_y = b$. By (typ-right-subterm), $U \to V > V$ and $W > B$. Hence, by transitivity, $U \to V > B$ and $\lambda xv \succ_\tau b$. Therefore, since $|w| >_{\mathbb{N}} |b|$, by induction hypothesis, b is computable.
- $(\lambda\mathcal{X})$ $w \in \varnothing$. Impossible.
- $(\lambda\eta)$ $v = ax$, $a \succeq w$ and $x \notin \mathrm{FV}(a)$. Since $\tau(\lambda xv) = \tau(a)$, we have $a \succeq_\tau w$. Let $u : U$ computable. We have $au = v^u_x$ computable by assumption. Therefore, a is computable. By Lemma 6.14, $[\![U \to V]\!]$ satisfies (comp-red) since $[\![U]\!] \neq \varnothing$ and $[\![V]\!]$ satisfies (comp-red). Therefore, w is computable too. □

Corollary 6.19. $[\![U \to V]\!]$ *satisfies* (comp-lam) *if:*

- $[\![U]\!]$ *satisfies* (comp-sn), (comp-red) *and* (comp-neutral)*;*
- $[\![V]\!]$ *satisfies* (comp-sn), (comp-red) *and* (comp-neutral)*;*
- $[\![V']\!]$ *satisfies* (comp-lam) *whenever* $V' \leq V$.

In conclusion, we can see that $[\![U \to V]\!]_I$ is a computability predicate if so are $[\![U]\!]_I$ and $[\![V']\!]_I$ for all $V' \leq V$. Therefore, if we can define a base type interpretation I so that, for every sort A, $I(\mathsf{A})$ is a computability predicate then, for all type T, $[\![T]\!]_I$ will be a computability predicate.

6.4. **Well-foundedness of core CPO.** We now define a set of neutral terms $\mathcal{N}$ and a base type interpretation I for proving the well-foundedness of core CPO.

Definition 6.20 (Neutral terms for core CPO)**.** Let $\mathcal{N}$ be the smallest set of terms containing the terms of the form $\mathsf{f}(\vec{t})$ and closed by (neutral-var), (neutral-beta) and (neutral-app).

One can easily check that $\mathcal{N}$ satisfies all the properties of Definition 6.10.

In contrast with the usual practice, but as in [59], our interpretation of sorts is not the set of strongly normalizing terms of that sort. To define the base type interpretation I, we proceed by induction on $\gtrdot$ which is well-founded by (typ-sn). So, let A be a sort and assume that I is defined for all sorts $\mathsf{B} \lessdot \mathsf{A}$. Then, let $I(\mathsf{A})$ be the least fixpoint of the monotone function F_{A} defined as follows:

$$F_{\mathsf{A}}(S) = \{t \in \mathcal{L} \mid t : \mathsf{A} \wedge (\forall u)(\forall U)\ t \succ_\tau u \wedge u : U \Rightarrow u \in [\![U]\!]_{I \cup \{(\mathsf{A},S)\}}\}$$

We now prove that F_{A} is indeed well-defined and monotone. Then, by Knaster and Tarski's fixpoint theorem [85], F_{A} admits a (least) fixpoint.

Lemma 6.21. *F_{A} is well-defined.*

Proof. The recursive call to $[\![U]\!]_{I \cup \{(\mathsf{A},S)\}}$ is well-defined because, by definition of $\succ_\tau$, we have $\mathsf{A} \geq U$. Hence, by Lemma 2.5, $\mathrm{Sort}_{\leqslant \mathsf{A}}(U)$. □

Lemma 6.22. *F_{A} is monotone.*

Proof. Let $S \subseteq S'$, $J = I \cup \{(\mathsf{A}, S)\}$, $J' = I \cup \{(\mathsf{A}, S')\}$ and $t \in F_{\mathsf{A}}(S)$. Then, (1) $t : \mathsf{A}$ and (2) $(\forall u)(\forall U)\ t \succ_\tau u \wedge u : U \Rightarrow u \in [\![U]\!]_J$. Now, we have $t \in F_{\mathsf{A}}(S')$ because t satisfies (1) and (2) with S replaced by S'. Indeed, assume that $t \succ_\tau u$ and $u : U$. By (2), $u \in [\![U]\!]_J$. By definition of $\succ_\tau$, $\mathsf{A} \geq U$. If $\mathsf{A} = U$, then $u \in [\![U]\!]_{J'}$ since $u \in [\![U]\!]_J = S \subseteq S' = [\![U]\!]_{J'}$. Otherwise, by Lemma 2.5, $\mathrm{Sort}_{\lessdot \mathsf{A}}(U)$. Therefore, by Lemma 6.8, $[\![U]\!]_J = [\![U]\!]_{J'}$ and $u \in [\![U]\!]_{J'}$. □

Note that the least fixpoint of F_{A} can be reached by transfinite iteration of F_{A} from $\varnothing$ [67, 30], that is, there is an ordinal $\mathfrak{a}$, such that $I(\mathsf{A}) = F_{\mathsf{A}}^{\mathfrak{a}}(\varnothing)$ where:

- $F_{\mathsf{A}}^0(S) = S$
- $F_{\mathsf{A}}^{\mathfrak{a}+1}(S) = F_{\mathsf{A}}(F_{\mathsf{A}}^{\mathfrak{a}}(S))$
- $F_{\mathsf{A}}^{\mathfrak{a}}(S) = \bigcup_{\mathfrak{b}<\mathfrak{a}} F_{\mathsf{A}}^{\mathfrak{b}}(S)$ if $\mathfrak{a}$ is a limit ordinal

We now check that type interpretations are computability predicates.

Lemma 6.23. *Given a sort A, $[\![\mathsf{A}]\!]$ satisfies* (comp-red)*,* (comp-neutral) *and* (comp-lam)*.*

Proof. We show each property in turn.

- (comp-red) Let $t \in [\![\mathsf{A}]\!]$ and assume that $t \succ_\tau u$ and $u : U$. Since $[\![\mathsf{A}]\!] = F_{\mathsf{A}}([\![\mathsf{A}]\!])$, we have $u \in [\![U]\!]$ by definition of F_{A}.
- (comp-neutral) Let $t : \mathsf{A}$ be a neutral term whose $\succ_\tau$-reducts are all computable. Since $[\![\mathsf{A}]\!] = F_{\mathsf{A}}([\![\mathsf{A}]\!])$, we have $t \in [\![\mathsf{A}]\!]$ by definition of F_{A}.
- (comp-lam) Trivial for typing reasons. □

Lemma 6.24. *Given a sort A, $[\![\mathsf{A}]\!]$ satisfies* (comp-sn) *if, for all type $U < \mathsf{A}$, $[\![U]\!]$ satisfies* (comp-sn)*.*

Proof. As already mentioned, $[\![\mathsf{A}]\!] = F_{\mathsf{A}}^{\mathfrak{a}}(\varnothing)$ for some ordinal $\mathfrak{a}$. Since $\varnothing$ satisfies (comp-sn), it therefore suffices to check that F_{A} preserves termination: if $S \subseteq \mathrm{SN}(\succ_\tau)$, then $F_{\mathsf{A}}(S) \subseteq \mathrm{SN}(\succ_\tau)$. So, let $S \subseteq \mathrm{SN}(\succ_\tau)$ and let $t \in F_{\mathsf{A}}(S)$. By definition of F_{A}, we have $t : \mathsf{A}$ and, if $t \succ_\tau u$ and $u : U$, then $u \in [\![U]\!]_J$ where $J = I \cup \{(\mathsf{A}, S)\}$. By definition of $\succ_\tau$, $\mathsf{A} \geq U$. If $\mathsf{A} = U$, then $[\![U]\!]_J = S$ and $u \in \mathrm{SN}(\succ_\tau)$ since $S \subseteq \mathrm{SN}(\succ_\tau)$. Otherwise, $u \in \mathrm{SN}(\succ_\tau)$ since $[\![U]\!] \subseteq \mathrm{SN}(\succ_\tau)$ by assumption. □

Theorem 6.25. *For all type T, $[\![T]\!]$ is a computability predicates,* i.e. *satisfies* (comp-sn)*,* (comp-red)*,* (comp-neutral) *and* (comp-lam)*.*

Proof. We proceed by induction on $\gtrdot$ which is well-founded by assumption (typ-sn). If T is a sort, then we can conclude by Lemma 6.23, Lemma 6.24 and induction hypothesis. Otherwise, $T = U \to V$. Since $T \rhd_l U$, by induction hypothesis, $[\![U]\!]$ is a computability predicate. Let now V' be a type such that $V \geq V'$. By (typ-right-subterm) and transitivity, $T > V'$. Hence, by induction hypothesis, $[\![V']\!]$ is a computability predicate. Therefore, $[\![U \to V]\!]$ satisfies (comp-sn) by Lemma 6.13, (comp-red) by Lemma 6.14, (comp-neutral) by Corollary 6.17 and (comp-lam) by Corollary 6.19. □

Now, we are left to prove that every function symbol is computable.

Lemma 6.26. *Let $\mathsf{f}^{|\vec{T}|} : \vec{T} \to U$ and $\vec{t} \in [\![\vec{T}]\!]$. Then, $\mathsf{f}(\vec{t}) \in [\![U]\!]$.*

Proof. By Theorem 6.25, $[\![\vec{T}]\!]$ satisfies (comp-sn). Hence, by Lemma 6.12, $(>_{\mathcal{F}}, (>_\tau)_{\mathrm{stat}})_{\mathrm{lex}}$ is well-founded. We now prove that, for all $(\mathsf{f}, \vec{t}) \in \Sigma$, $\mathsf{f}(\vec{t})$ is computable, by induction on $(>_{\mathcal{F}}, (>_\tau)_{\mathrm{stat}})_{\mathrm{lex}}$ (1).

First, we check that $t = \mathsf{f}(\vec{t})$ is computable if all its $>_\tau$-reducts so are. This follows from the facts that, by definition of $\mathcal{N}$, t is neutral and, by Theorem 6.25, $[\![U]\!]$ satisfies (comp-neutral).

We now prove that, for all finite sets of variables X, for all substitutions σ such that $\mathrm{dom}(\sigma) \cap \mathrm{FV}(\vec{t}) = \varnothing$ and σ is computable on X, and for all terms w such that $\mathsf{f}(\vec{t}) >^X w$, we have $w\sigma$ computable, by induction on the size of w (2). Note that $\vec{t}\sigma = \vec{t}$ since $\mathrm{dom}(\sigma) \cap \mathrm{FV}(\vec{t}) = \varnothing$.

- ($\mathcal{F}_b\rhd$) $(\exists i)\ t_i \geq_\tau w$. By stability by substitution of $\geq_\tau$ (Lemma 1), we have $t_i\sigma \geq_\tau w\sigma$. Therefore, $w\sigma$ is computable since, by Theorem 6.25, $[\![V]\!]$ satisfies (comp-red).
- ($\mathcal{F}_b{=}$) There are g and $\vec{u}$ such that $w = \mathsf{g}(\vec{u})$, $\mathsf{f} \simeq_{\mathcal{F}} \mathsf{g}$, $(\forall i)\ \mathsf{f}(\vec{t}) >^X u_i$ and $\vec{t}\,(>_\tau)_{\mathrm{stat}(\mathsf{f})}\,\vec{u}$. Since $(\forall i)\ \mathsf{f}(\vec{t}) >^X u_i$, by induction hypothesis (2), $\vec{u}\sigma$ are computable. If $t_i >_\tau u_j$ then, by stability by substitution (Lemma 1), $t_i = t_i\sigma >_\tau u_j\sigma$. Therefore, $\vec{t}\,(>_\tau)_{\mathrm{stat}(\mathsf{f})}\,\vec{u}\sigma$ and, by induction hypothesis (1), $\mathsf{g}(\vec{u})\sigma$ is computable.
- ($\mathcal{F}_b{>}$) Since $\mathsf{f}(\vec{t}) >^X \vec{u}$, by induction hypothesis (2), $\vec{u}\sigma$ are computable. Hence, $\mathsf{g}(\vec{u})\sigma$ is computable by induction hypothesis (1).
- ($\mathcal{F}_b@$) Since $\mathsf{f}(\vec{t}) >^X u$ and $\mathsf{f}(\vec{t}) >^X v$, by induction hypothesis (2), $u\sigma$ and $v\sigma$ are computable. Therefore, $(uv)\sigma = (u\sigma)(v\sigma)$ is computable.
- ($\mathcal{F}_b\lambda$) Wlog we can assume that σ is away from $\{y\}$ and $y \notin \mathrm{FV}(\mathsf{f}(\vec{t}))$. Hence, $(\lambda yv)\sigma = \lambda y(v\sigma)$. By Theorem 6.25, $\lambda y(v\sigma)$ is computable if, for all computable $u : \tau(y)$, $(v\sigma)^u_y$ is computable. Since σ is away from $\{y\}$, $(v\sigma)^u_y = (v^z_y)\theta$ where $\theta = \sigma \cup \{(z, u)\}$. Let $Z = X \cup \{z\}$. Since $\mathsf{f}(\vec{t}) >^Z v^z_y$, $\mathrm{dom}(\theta) \cap \mathrm{FV}(\mathsf{f}(\vec{t})) = \varnothing$ and θ is computable on Z, we have $v^z_y\theta$ computable by induction hypothesis (2).
- ($\mathcal{F}_b\mathcal{X}$) $w \in X$. Then, $w\sigma$ is computable since σ is computable on X. □

Theorem 6.27. *The relation $>_\tau$ of Definition 5.1 is well-founded.*

Proof. After Theorem 6.11, Theorem 6.25 and Lemma 6.26. □

We can therefore conclude that $>_\tau^+$ is a monotone, stable, well-founded order.

The well-foundedness proof of core CPO is actually similar to that of HORPO, although the proof here is presented in a quite different style from HORPO's monolithic proof [59]. This similarity fades away with the two coming extensions, to inductive types and to small

symbols. The reason why we have split the proof into small pieces is indeed to factor out its structure and those parts which are common to core CPO and its extensions.

7. Accessibility

In this section, we introduce an extension of the core definition that will allow us to handle rewrite rules like the ones defining recursors for strictly positive inductive types as used in the Coq proof assistant [29, 54].

Example 7.1. Consider the inductive type O of "Brouwer ordinals" whose constructors are $\mathsf{zero} : \mathsf{O}$ for zero, $\mathsf{suc}^1 : \mathsf{O} \to \mathsf{O}$ for successor, and $\mathsf{lim}^1 : (\mathsf{N} \to \mathsf{O}) \to \mathsf{O}$ for limit, where N is the inductive type of Peano (unary) natural numbers with constructors $\mathsf{0} : \mathsf{N}$ and $\mathsf{s}^1 : \mathsf{N} \to \mathsf{N}$. Given a type A, the recursor (of arity 4) at type A

$$\mathsf{rec}_{\mathsf{O}}^{A} : \mathsf{O} \to A \to (\mathsf{O} \to A \to A) \to ((\mathsf{N} \to \mathsf{O}) \to (\mathsf{N} \to A) \to A) \to A$$

can be defined by the following rewrite rules:

$$\begin{aligned}
\mathsf{rec}_{\mathsf{O}}^{A}(\mathsf{zero}, u, v, w) &\to u\\
\mathsf{rec}_{\mathsf{O}}^{A}(\mathsf{suc}(x), u, v, w) &\to v\, x\, (\mathsf{rec}_{\mathsf{O}}^{A}(x, u, v, w))\\
\mathsf{rec}_{\mathsf{O}}^{A}(\mathsf{lim}(y), u, v, w) &\to w\, y\, (\lambda n\, \mathsf{rec}_{\mathsf{O}}^{A}(y\, n, u, v, w))
\end{aligned}$$

To capture such a relation, we need the following two comparisons to succeed:

$$\mathsf{rec}_{\mathsf{O}}^{A}(\mathsf{lim}(y), u, v, w) \succ y \quad \text{and} \quad \mathsf{lim}(y) \succ_\tau y\, n.$$

The second comparison cannot succeed unless we allow non empty sets X of variables in $(\mathcal{F}_b{=})$ in order to have $\mathsf{lim}(y) \succ_\tau^{\{n\}} y\, n$, but we have seen in Section 5.3 that this may lead to non-termination. Instead, we will use a specific ordering: the *structural ordering* introduced by Coquand for dealing with such kind of definitions in the calculus of constructions [28].

For the first comparison to succeed, since the type of y is bigger than the type of $\mathsf{lim}(y)$, we must *not* check types in $(\mathcal{F}_b{\triangleright})$, but we have seen in Section 5.3 that this may lead to non-termination. To solve this problem, we will compare in $(\mathcal{F}_b{\triangleright})$ the right-hand side with possibly deep subterms of the left-hand side.

We cannot take any deep subterm however, as shown by the following example: assuming the signature $\mathsf{c}^1 : (\mathsf{A} \to \mathsf{B}) \to \mathsf{A}$ and $\mathsf{f}^1 : \mathsf{A} \to (\mathsf{A} \to \mathsf{B})$, the deep subterm comparison $\mathsf{f}(\mathsf{c}(x)) \succ_\tau x$ leads to non-termination, since, taking $t = \lambda x\, \mathsf{f}(x)\, x$, we have $\mathsf{f}(\mathsf{c}(t))\, \mathsf{c}(t) \succ_\tau t\, \mathsf{c}(t) \succ_\tau \mathsf{f}(\mathsf{c}(t))\, \mathsf{c}(t)$ by monotony and $(@\beta)$. There are two cases where deep subterms can be used: as for the first, deep subterms whose type is a (sufficiently small) sort [59]; as for the second, Mendler showed that pattern-matching on constructors of a sort A having an argument whose type has a negative occurrence of A wrt the arrow type constructor (see Definition 7.2 just after), leads to non-termination, while, on the contrary, β-reduction combined with recursion combinators for positive inductive types terminates [73, 74].

7.1. Accessible subterms.

In this sub-section, we first define the sets of positive and negative positions of a type, and the notions of accessible and structurally smaller term, before we prove some properties of these notions.

Definition 7.2 (Positive and negative positions in a type)**.** The sets $\mathrm{Pos}(T)$, $\mathrm{Pos}(\mathsf{A}, T)$, $\mathrm{Pos}^+(T)$ and $\mathrm{Pos}^-(T)$ of positions, positions of A, positive positions and negative positions in a type T are inductively defined as follows:

- $\mathrm{Pos}(\mathsf{A}) = \mathrm{Pos}^+(\mathsf{A}) = \mathrm{Pos}(\mathsf{A}, \mathsf{A}) = \{\varepsilon\}$

- $\mathrm{Pos}^-(\mathsf{A}) = \varnothing$
- $\mathrm{Pos}(\mathsf{A},\mathsf{B}) = \varnothing$ if $\mathsf{A} \neq \mathsf{B}$
- $\mathrm{Pos}(T \to U) = \{1p \mid p \in \mathrm{Pos}(T)\} \cup \{2p \mid p \in \mathrm{Pos}(U)\}$
- $\mathrm{Pos}(\mathsf{A}, T \to U) = \{1p \mid p \in \mathrm{Pos}(\mathsf{A},T)\} \cup \{2p \mid p \in \mathrm{Pos}(\mathsf{A},U)\}$
- $\mathrm{Pos}^+(T \to U) = \{1p \mid p \in \mathrm{Pos}^-(T)\} \cup \{2p \mid p \in \mathrm{Pos}^+(U)\}$
- $\mathrm{Pos}^-(T \to U) = \{1p \mid p \in \mathrm{Pos}^+(T)\} \cup \{2p \mid p \in \mathrm{Pos}^-(U)\}$

A sort A *occurs only positively* (resp. *negatively*) in T if $\mathrm{Pos}(\mathsf{A},T) \subseteq \mathrm{Pos}^+(T)$ (resp. $\mathrm{Pos}(\mathsf{A},T) \subseteq \mathrm{Pos}^-(T)$).

For instance, for $T = (\mathsf{A} \to \mathsf{B}) \to \mathsf{B}$ with $\mathsf{A} \neq \mathsf{B}$, we have $\mathrm{Pos}(T) = \{\varepsilon, 1, 2, 11, 12\}$, $\mathrm{Pos}^+(T) = \{11, 2\}$, $\mathrm{Pos}^-(T) = \{12\}$, $\mathrm{Pos}(\mathsf{A},T) = \{11\}$ and $\mathrm{Pos}(\mathsf{B},T) = \{12, 2\}$. Hence, A occurs only positively in T, but B has both positive and negative occurrences in T.

Definition 7.3 (Accessible arguments)**.** For every $\mathsf{f}^{\alpha(\mathsf{f})} : \vec{T} \to \mathsf{A}$, we assume given a set $\mathrm{Acc}(\mathsf{f})$ of *accessible* arguments of f such that $i \in \mathrm{Acc}(\mathsf{f})$ implies $\mathrm{Sort}_{\leqslant\mathsf{A}}(T_i)$ and $\mathrm{Pos}(\mathsf{A},T_i) \subseteq \mathrm{Pos}^+(T_i)$.

Note that, if $\alpha(\mathsf{f}) < |\vec{T}|$, the output type of f is functional.

Let us consider Example 7.1 and assume that $\mathsf{O} > \mathsf{N}$. Then, O occurs only positively in the type of the first argument of suc, and we can take $\mathrm{Acc}(\mathsf{suc}) = \{1\}$. Similarly, we can take $\mathrm{Acc}(\mathsf{lim}) = \{1\}$.

We can now introduce those sorts A which are not bigger than any arrow type and will be interpreted by $\mathrm{SN}_{\mathsf{A}}(>_\tau)$ later:

Definition 7.4 (Basic sorts)**.** A sort A is *basic* if, for all type $T < \mathsf{A}$, T is a basic sort and, for all $\mathsf{f}^{\alpha(\mathsf{f})} : \vec{T} \to \mathsf{A}$ and $i \in \mathrm{Acc}(\mathsf{f})$, $T_i = \mathsf{A}$ or T_i is a basic sort.

In particular, are basic all first-order data types like unary natural numbers, lists, trees, etc. whose constructors do not take a function as argument.

Accessibility blends accessible arguments and subterms of basic sort:

Definition 7.5 (Accessibility)**.** A term u is said to be *accessible* in a term t if:

- u is a subterm of basic sort of t such that $\mathrm{FV}(u) \subseteq \mathrm{FV}(t)$, written $t \triangleright_b^s u$, or
- there are $\mathsf{f}^{\alpha(\mathsf{f})} : \vec{T} \to \mathsf{A}$, $\vec{t} : \vec{T}$ and $i \in \mathrm{Acc}(\mathsf{f})$ such that $t = \mathsf{f}(t_1, \ldots, t_{\alpha(\mathsf{f})}) t_{\alpha(\mathsf{f})+1} \ldots t_{|\vec{T}|}$ and $t_i \trianglerighteq_a u$, written $t \triangleright_a u$,

where $\trianglerighteq_b^s$ and $\trianglerighteq_a$ are the reflexive closures of $\triangleright_b^s$ and $\triangleright_a$ respectively.

Coming back to Example 7.1, we have x accessible in $\mathsf{suc}(x)$ since $\mathsf{suc}(x) \triangleright_a x$, and y accessible in $\mathsf{lim}(y)$ since $\mathsf{lim}(y) \triangleright_a y$.

Lemma 7.6. *If $t \triangleright_a u : U$, then there are two sorts A and B such that $t : \mathsf{A}$, $\mathsf{B} \leqslant \mathsf{A}$, $\mathrm{Sort}_{\leqslant\mathsf{B}}(U)$ and $\mathrm{Pos}(\mathsf{B},U) \subseteq \mathrm{Pos}^+(U)$.*

Proof. By induction on $\triangleright_a$, which is clearly well-founded. Assume that there are $\mathsf{f}^{\alpha(\mathsf{f})} : \vec{T} \to \mathsf{A}$, $\vec{t} : \vec{T}$ and $i \in \mathrm{Acc}(\mathsf{f})$ such that $t = \mathsf{f}(t_1, \ldots, t_{\alpha(\mathsf{f})}) t_{\alpha(\mathsf{f})+1} \ldots t_{|\vec{T}|}$ and $t_i \trianglerighteq_a u$. By definition of Acc, $\mathrm{Sort}_{\leqslant\mathsf{A}}(T_i)$ and $\mathrm{Pos}(\mathsf{A},T_i) \subseteq \mathrm{Pos}^+(T_i)$. If $t_i = u$ then $U = T_i$ and we are done with $\mathsf{B} = \mathsf{A}$. Assume now that $t_i \triangleright_a u$. Then, by induction hypothesis, there are two sorts B and C such that $t_i : \mathsf{B}$, $\mathsf{C} \leqslant \mathsf{B}$, $\mathrm{Sort}_{\leqslant\mathsf{C}}(U)$ and $\mathrm{Pos}(\mathsf{C},U) \subseteq \mathrm{Pos}^+(U)$. Since $\mathrm{Sort}_{\leqslant\mathsf{A}}(T_i)$ and $T_i = \mathsf{B}$, we have $\mathsf{B} \leqslant \mathsf{A}$. By transitivity, we get $\mathsf{C} \leqslant \mathsf{A}$. □

Corollary 7.7. *If* $t : \mathsf{A}$, $t \triangleright_a u : U$ *and* A *occurs in* U, *then* $\mathrm{Sort}_{\leqslant\mathsf{A}}(U)$ *and* $\mathrm{Pos}(\mathsf{A},U) \subseteq \mathrm{Pos}^+(U)$.

Proof. By Lemma 7.6, there is a sort $\mathsf{B} \leqslant \mathsf{A}$ such that $\mathrm{Sort}_{\leqslant\mathsf{B}}(U)$ and $\mathrm{Pos}(\mathsf{B},U) \subseteq \mathrm{Pos}^+(U)$. Since A occurs in U, $\mathsf{A} \leqslant \mathsf{B}$. Therefore, $\mathsf{B} = \mathsf{A}$ and we are done. □

Definition 7.8 ([28])**.** Given a finite set X of variables, we say that u is *structurally smaller than* t wrt X, written $t \triangleright_{@}^{X} u$, if there are A, v and $\vec{x} : \vec{U}$ such that $t : \mathsf{A}$, $u : \mathsf{A}$, $u = v\vec{x}$, $t \triangleright_a v$, $\vec{x} \in X$ and $\mathrm{Pos}(\mathsf{A},\vec{U}) = \varnothing$.

One can easily check:

Lemma 7.9. $\triangleright_b^s$ *and* $\triangleright_a$ *(resp.* $\triangleright_{@}^{X}$*) are stable by substitution (resp. away from* X*).*

7.2. **CPO with accessible subterms.**

Definition 7.10 (CPO with accessible subterms)**.** The relation $\succ^X$ is extended by replacing the rules $(\mathcal{F}_b\triangleright)$ and $(\mathcal{F}_b{=})$ of Figure 1 by the ones of Figure 2.

Figure 2: New CPO rules with accessible subterms

$(\mathcal{F}_b\triangleright)$	$\mathsf{f}(\vec{t}) \succ^X v$ if $\mathsf{f} \in \mathcal{F}_b$ and $\vec{t}\ \trianglerighteq_b^s \trianglerighteq_a \geq_\tau v$
$(\mathcal{F}_b{=})$	$\mathsf{f}(\vec{t}) \succ^X \mathsf{g}(\vec{u})$ if $\mathsf{f} \in \mathcal{F}_b$, $\mathsf{f} \simeq_{\mathcal{F}} \mathsf{g}$, $\mathsf{f}(\vec{t}) \succ^X \vec{u}$ and $\vec{t}\,(\succ_\tau \cup \triangleright_{@}^{X} \geq_\tau)_{\mathrm{stat}(\mathsf{f})}\,\vec{u}$

The rules of Example 7.1 are now easily oriented by CPO. Take for instance the third rule. It is included in CPO since, by $(\mathcal{F}_b@)$:

- $l = \mathsf{rec}_{\mathsf{O}}^A(\mathsf{lim}(y), u, v, w) \succ w\,x$ by $(\mathcal{F}_b@)$since:
  - $l > w$ by $(\mathcal{F}_b\triangleright)$,
  - $l > y$ by $(\mathcal{F}_b\triangleright)$ since $\mathsf{lim}(y) \triangleright_a y$,
- $l \succ \lambda n\,\mathsf{rec}_{\mathsf{O}}^A(y\,n, u, v, w)$ by $(\mathcal{F}_b\lambda)$ since $l \succ^{\{n\}} \mathsf{rec}_{\mathsf{O}}^A(y\,n, u, v, w)$ because, by $(\mathcal{F}_b{=})$:
  - $l \succ^{\{n\}} y\,n$ since by $(\mathcal{F}_b@)$:
    - $l > y$ as already seen,
    - $l > n$ by $(\mathcal{F}_b\mathcal{X})$,
  - $l > u, v, w$ by $(\mathcal{F}_b\triangleright)$,
  - $\mathsf{lim}(y) \triangleright_{@}^{\{n\}} y\,n$.

Following [12], we could strengthen CPO further by defining $\triangleright_{@}^{X}$ and $\succ^X$ simultaneously, by replacing in $(\mathcal{F}_b{=})$, $\triangleright_{@}^{X}$ by $\triangleright_{@}^{\mathsf{f}(\vec{t}),X}$ and, in the definition of $\triangleright_{@}^{X}$, $\vec{x} \in X$ by $\mathsf{f}(\vec{t}) \succ^X \vec{x}$.

7.3. **Comparison with CHORPO.**

CHORPO is a variant of HORPO which was also aiming at ordering recursors of inductive types like Brouwer's ordinals. In rules (1), (3), (4) and (7) of the 12 rules of HORPO as recalled in Section 5.4, one has to show recursively that every direct subterm of the left-hand side $\mathsf{f}(\vec{t})$ is bigger than (or equal to) the right-hand side. In CHORPO, one can also use in addition to the direct subterms, any term of the *computability closure* $\mathrm{CC}(\mathsf{f}(\vec{t}), \varnothing)$ of the left-hand side, a set inductively defined by 6 rules (CC1) to (CC6) that, for most of them, correspond to CPO rules as follows.

$\mathrm{CC}(\mathsf{f}(\vec{t}), X)$ must contain $\{\vec{t}\}$, which corresponds to $(\mathcal{F}_b\triangleright)$, and X, which corresponds to $(\mathcal{F}_b\mathcal{X})$; (CC1) says that $\mathrm{CC}(\mathsf{f}(\vec{t}), \varnothing)$ contains any term u of minimal type such that $\vec{t} \triangleright^s u$,

where $t \triangleright^s u$ if $t \triangleright u$ and $\mathrm{FV}(u) \subseteq \mathrm{FV}(t)$, which corresponds to $(\mathcal{F}_b\triangleright)$, $(@\triangleright)$ and $(\lambda\triangleright)$; (CC2) corresponds to $(\mathcal{F}_b{>})$; (CC3) corresponds to $(\mathcal{F}_b{=})$ with $\succ_\tau$ replaced by $\succ_\tau \cup \triangleright^s_\tau$; (CC4) corresponds to $(\mathcal{F}_b@)$ and (CC5) to $(\mathcal{F}_b\lambda)$.

On the other hand, (CC6) says that $\mathrm{CC}(\mathsf{f}(\vec{t}), \varnothing)$ is closed by $\succ_{\mathrm{horpo}}$. Capturing such a rule in CPO requires to consider the transitive closure of $\succ_\tau$ in $(\mathcal{F}_b{=})$ which would most presumably turn CPO into an undecidable relation, as it is probably already the case of CHORPO for the same reason.

In conclusion, while CHORPO and CPO look incomparable, the restriction of CHORPO to (CC1), (CC2), (CC3), (CC4), (CC5) is included in CPO. In fact, CPO can be seen as a decidable reformulation of CHORPO integrating in a simple, uniform and more powerful way both HORPO and the notion of computability closure. Note finally that HORPO and the computability closure are themselves already related, as shown in [12]. More precisely, the first version of HORPO [58] is included in the fixpoint of the monotone function mapping $>$ to the relation $>^{\varnothing}_{\mathrm{CC}}$ such that $t >^{X}_{\mathrm{CC}} u$ if $u \in \mathrm{CC}(t, X)$, RPO being equal to this fixpoint when restricted to first-order terms.

7.4. **Computability with accessible subterms.**

Lemma 7.11 (Basic properties)**.**

- $\succ^X$ *is well-defined.*
- $\succ_\tau$ *is monotone.*
- *If* $a \succ^X b$*, then* $\mathrm{FV}(b) \subseteq \mathrm{FV}(a) \cup X$*.*
- $\succ^X$ *is stable by* α*-equivalence.*
- $\succ^X$ *is stable by substitution away from* X*.*
- *If* $e, e' \in \mathcal{X}$*,* $\tau(e) = \tau(e')$*,* $t \succ^X u$ *and* $e' \notin \mathrm{FV}(\lambda e u)$*, then* $t \succ^{X-\{e\}\cup\{e'\}} u^{e'}_e$*.*

Proof. As for the core definition using Lemma 7.9 and the fact that, if $a \triangleright_a b$, then $\mathrm{FV}(b) \subseteq \mathrm{FV}(a)$. □

In order to extend the well-founded proof of core CPO to accessible subterms, we need to define a set of neutral terms and a base type interpretation so that accessible arguments of a computable term $\mathsf{f}(\vec{t})$ are computable. Hence, the following definitions:

Definition 7.12 (Neutral terms for CPO with accessible subterms)**.** Let $\mathcal{N}$ be the smallest set of terms containing the terms of the form $\mathsf{f}(\vec{t})$ with $\mathrm{Acc}(\mathsf{f}) = \varnothing$, and closed by (neutral-var), (neutral-beta) and (neutral-app).

One can easily check that $\mathcal{N}$ satisfies all the properties of Definition 6.10. Note that, now, a term is neutral if and only if it is of the form $x\,\vec{v}$, $(\lambda x a)\,b\,\vec{v}$, or $\mathsf{f}(\vec{t})\,\vec{v}$ with $\mathrm{Acc}(\mathsf{f}) = \varnothing$.

To define the base type interpretation I, we proceed as for core CPO by well-founded induction on $\gtrdot$. So, let A be a sort and assume that I is defined for all sorts $\mathsf{B} \lessdot \mathsf{A}$. Then, let $I(\mathsf{A})$ be the least fixpoint of the monotone function F_{A} defined as follows:

$$\begin{aligned} F_{\mathsf{A}}(S) = \{t \in \mathcal{L} \mid t : \mathsf{A} \wedge (\forall u)(\forall U)\ t \succ_\tau u \wedge u : U \Rightarrow u \in [\![U]\!]_{I\cup\{(\mathsf{A},S)\}} \\ \wedge\ (\forall \mathsf{f})(\forall \vec{T})(\forall \vec{t})(\forall i)\ \mathsf{f}^{\alpha(\mathsf{f})} : \vec{T} \to \mathsf{A} \wedge t = \mathsf{f}(t_1, \ldots, t_{\alpha(\mathsf{f})}) t_{\alpha(\mathsf{f})+1} \ldots t_{|\vec{T}|} \wedge i \in \mathrm{Acc}(\mathsf{f}) \\ \Rightarrow t_i \in [\![T_i]\!]_{I\cup\{(\mathsf{A},S)\}}\} \end{aligned}$$

Note that, by this definition, a term $\mathsf{f}(t_1,\dots,t_{\alpha(\mathsf{f})})t_{\alpha(\mathsf{f})+1}\dots t_n : \mathsf{A}$ is computable if all its $\succ_\tau$-reducts and all its accessible arguments t_i with $i \in \mathrm{Acc}(\mathsf{f})$ are computable. This makes the terms of this form behave like neutral terms when $\vec{t}$ are computable.

We now prove that F_A is indeed well-defined and monotone.

Lemma 7.13. *F_A is well-defined.*

Proof. The calls to $[\![U]\!]_{I\cup\{(\mathsf{A},S)\}}$ and $[\![T_i]\!]_{I\cup\{(\mathsf{A},S)\}}$ with $i \in \mathrm{Acc}(\mathsf{f})$ are well-defined because every sort occurring in U or T_i is $\leqslant$ to A. Indeed, by definition of $\succ_\tau$, we have $\mathsf{A} \geqslant U$. Hence, by Lemma 2.5, $\mathrm{Sort}_{\leqslant\mathsf{A}}(U)$. As for T_i, it follows by definition of $\mathrm{Acc}(\mathsf{f})$. □

Lemma 7.14. *Let T be a type such that $\mathrm{Sort}_{\leqslant\mathsf{A}}(T)$. Then, the function $S \mapsto [\![T]\!]_{I\cup\{(\mathsf{A},S)\}}$ is monotone (resp. anti-monotone) wrt set inclusion if $\mathrm{Pos}(\mathsf{A},T) \subseteq \mathrm{Pos}^+(T)$ (resp. $\mathrm{Pos}(\mathsf{A},T) \subseteq \mathrm{Pos}^-(T)$).*

Proof. Let $S \subseteq S'$, $J = I \cup \{(\mathsf{A},S)\}$ and $J' = I \cup \{(\mathsf{A},S')\}$. We proceed by induction on T.

- $T = \mathsf{A}$. Then, $[\![T]\!]_J = S \subseteq S' = [\![T]\!]_{J'}$.
- $T = \mathsf{B} \lessdot \mathsf{A}$. Then, $[\![T]\!]_J = I(\mathsf{B}) = [\![T]\!]_{J'}$.
- $T = U \to V$ and $\mathrm{Pos}(\mathsf{A},T) \subseteq \mathrm{Pos}^+(T)$. Let $t \in [\![T]\!]_J$. By definition of $[\![T]\!]$, $t \in [\![T]\!]_{J'}$ if, for all $u \in [\![U]\!]_{J'}$, $tu \in [\![V]\!]_{J'}$. By definition, $\mathrm{Pos}(\mathsf{A},T) = \{1p \mid p \in \mathrm{Pos}(\mathsf{A},U)\} \cup \{2p \mid p \in \mathrm{Pos}(\mathsf{A},V)\}$ and $\mathrm{Pos}^+(T) = \{1p \mid p \in \mathrm{Pos}^-(U)\} \cup \{2p \mid p \in \mathrm{Pos}^+(V)\}$. Hence, $\mathrm{Pos}(\mathsf{A},U) \subseteq \mathrm{Pos}^-(U)$ and $\mathrm{Pos}(\mathsf{A},V) \subseteq \mathrm{Pos}^+(V)$. Therefore, by induction hypothesis, $[\![U]\!]_{J'} \subseteq [\![U]\!]_J$ and $[\![V]\!]_J \subseteq [\![V]\!]_{J'}$. So, $u \in [\![U]\!]_J$ and, since $t \in [\![T]\!]_J$, we have $tu \in [\![V]\!]_J \subseteq [\![V]\!]_{J'}$.
- $T = U \to V$ and $\mathrm{Pos}(\mathsf{A},T) \subseteq \mathrm{Pos}^-(T)$. Let $t \in [\![T]\!]_{J'}$. By definition of $[\![T]\!]$, $t \in [\![T]\!]_J$ if, for all $u \in [\![U]\!]_J$, $tu \in [\![V]\!]_J$. By definition, $\mathrm{Pos}(\mathsf{A},T) = \{1p \mid p \in \mathrm{Pos}(\mathsf{A},U)\} \cup \{2p \mid p \in \mathrm{Pos}(\mathsf{A},V)\}$ and $\mathrm{Pos}^-(T) = \{1p \mid p \in \mathrm{Pos}^+(U)\} \cup \{2p \mid p \in \mathrm{Pos}^-(V)\}$. Hence, $\mathrm{Pos}(\mathsf{A},U) \subseteq \mathrm{Pos}^+(U)$ and $\mathrm{Pos}(\mathsf{A},V) \subseteq \mathrm{Pos}^-(V)$. Therefore, by induction hypothesis, $[\![U]\!]_J \subseteq [\![U]\!]_{J'}$ and $[\![V]\!]_{J'} \subseteq [\![V]\!]_J$. So, $u \in [\![U]\!]_{J'}$ and, since $t \in [\![T]\!]_{J'}$, we have $tu \in [\![V]\!]_{J'} \subseteq [\![V]\!]_J$. □

Lemma 7.15. *F_A is monotone.*

Proof. Let $S \subseteq S'$, $J = I \cup \{(\mathsf{A},S)\}$, $J' = I \cup \{(\mathsf{A},S')\}$ and $t \in F_\mathsf{A}(S)$. Then, (1) $t : \mathsf{A}$, (2) $(\forall u)(\forall U)\ t \succ_\tau u \wedge u : U \Rightarrow u \in [\![U]\!]_J$, and (3) $(\forall \mathsf{f})(\forall \vec{T})(\forall \vec{t})(\forall i)\ \mathsf{f}^{\alpha(\mathsf{f})} : \vec{T} \to \mathsf{A} \wedge t = \mathsf{f}(t_1,\dots,t_{\alpha(\mathsf{f})})t_{\alpha(\mathsf{f})+1}\dots t_{|\vec{T}|} \wedge i \in \mathrm{Acc}(\mathsf{f}) \Rightarrow t_i \in [\![T_i]\!]_J$. We have $t \in F_\mathsf{A}(S')$ because t satisfies (1), (2) and (3) with S replaced by S':

(1) $t : \mathsf{A}$ by (1).
(2) Assume that $t \succ_\tau u$ and $u : U$. By (2), $u \in [\![U]\!]_J$. By definition of $\succ_\tau$, $\mathsf{A} \geqslant U$. If $\mathsf{A} = U$, then $u \in [\![U]\!]_{J'}$ since $u \in [\![U]\!]_J = S \subseteq S' = [\![U]\!]_{J'}$. Otherwise, by Lemma 2.5, $\mathrm{Sort}_{\lessdot\mathsf{A}}(U)$. Therefore, by Lemma 6.8, $[\![U]\!]_J = [\![U]\!]_{J'}$ and $u \in [\![U]\!]_{J'}$.
(3) Assume that $\mathsf{f}^{\alpha(\mathsf{f})} : \vec{T} \to \mathsf{A}$, $t = \mathsf{f}(t_1,\dots,t_{\alpha(\mathsf{f})})t_{\alpha(\mathsf{f})+1}\dots t_{|\vec{T}|}$ and $i \in \mathrm{Acc}(\mathsf{f})$. By definition of Acc, $\mathrm{Sort}_{\leqslant\mathsf{A}}(T_i)$, $\mathrm{Pos}(\mathsf{A},T_i) \subseteq \mathrm{Pos}^+(T_i)$ and, by Lemma 7.14, $[\![T_i]\!]_J \subseteq [\![T_i]\!]_{J'}$. Therefore, $t_i \in [\![T_i]\!]_{J'}$ since $t_i \in [\![T_i]\!]_J$ by (3). □

7.5. **Well-foundedness of the structural term ordering.**

Lemma 7.16. *The function $\mathfrak{a} \mapsto F_\mathsf{A}^\mathfrak{a}(\varnothing)$ is monotone.*

Proof. Let $J^\mathfrak{a} = F_\mathsf{A}^\mathfrak{a}(\varnothing)$. We prove by induction on $\mathfrak{b}$ that, for all $\mathfrak{a} < \mathfrak{b}$, $J^\mathfrak{a} \subseteq J^\mathfrak{b}$.

- $\mathfrak{b} = \mathfrak{c} + 1$. Then, $J^{\mathfrak{b}} = F_{\mathsf{A}}(J^{\mathfrak{c}})$. If $\mathfrak{a} = \mathfrak{c}$, then $J^{\mathfrak{a}} \subseteq J^{\mathfrak{b}}$ by definition of F_{A}. Otherwise, $\mathfrak{a} < \mathfrak{c}$ and, by induction hypothesis, $J^{\mathfrak{a}} \subseteq J^{\mathfrak{c}}$. By Lemma 7.15, $J^{\mathfrak{a}+1} \subseteq J^{\mathfrak{c}+1}$. Since $J^{\mathfrak{a}} \subseteq J^{\mathfrak{a}+1}$ by definition of F_{A}, we have $J^{\mathfrak{a}} \subseteq J^{\mathfrak{b}}$.
- $\mathfrak{b}$ is a limit ordinal. Then, $J^{\mathfrak{a}} \subseteq J^{\mathfrak{b}}$ by definition of $J^{\mathfrak{b}}$. □

The functions $F^{\mathfrak{a}}_{\mathsf{A}}$ provide us with a well-founded relation that is the basis of the well-foundedness of the structural term ordering when it is instantiated by computable terms:

Definition 7.17 (Rank ordering)**.** Let the rank of a term $t \in [\![\mathsf{A}]\!]$, $\mathrm{rk}_{\mathsf{A}}(t)$, be the smallest ordinal $\mathfrak{a}$ such that $t \in F^{\mathfrak{a}}_{\mathsf{A}}(\varnothing)$. Then, let $t \sqsupset u$ if there is a sort A such that $t \in [\![\mathsf{A}]\!]$, $u \in [\![\mathsf{A}]\!]$ and $\mathrm{rk}_{\mathsf{A}}(t) > \mathrm{rk}_{\mathsf{A}}(u)$.

We now prove that $\succ_\tau$ is included in $\sqsupset$ and that their union is strongly normalizing on computable terms.

Lemma 7.18. *If $t \in [\![\mathsf{A}]\!]$, $u \in [\![\mathsf{A}]\!]$ and $t \succ_\tau u$, then $t \sqsupset u$.*

Proof. By definition, we have $t \in F^{\mathfrak{a}}_{\mathsf{A}}(\varnothing)$ where $\mathfrak{a} = \mathrm{rk}_{\mathsf{A}}(t)$. We can neither have $\mathfrak{a} = 0$ nor $\mathfrak{a}$ be a limit ordinal. So, there is $\mathfrak{b}$ such that $\mathfrak{a} = \mathfrak{b} + 1$. Hence, $F^{\mathfrak{a}}_{\mathsf{A}}(\varnothing) = F_{\mathsf{A}}(F^{\mathfrak{b}}_{\mathsf{A}}(\varnothing))$ and $u \in F^{\mathfrak{b}}_{\mathsf{A}}(\varnothing)$ by definition of F_{A}. Therefore, $t \sqsupset u$. □

Lemma 7.19. *$[\![T]\!] \subseteq \mathrm{SN}(\succ_\tau \cup \sqsupset)$ if, for all $T' \leq T$, $[\![T']\!]$ satisfies* (comp-sn)*.*

Proof. Assume that there is an infinite $(\succ_\tau \cup \sqsupset)$-decreasing sequence $(t_i)_{i\geq 0}$ such that $t_0 \in [\![T]\!]$ and $t_i : T_i$. Then, $(T_i)_{i\geq 0}$ is an infinite $\geq$-decreasing sequence. Since $>$ is well-founded by (typ-sn), there must be some j such that, for all $i \geq j$, $T_i = T_j$. If T_j is a sort then, by Lemma 7.18, $(t_i)_{i\geq j}$ is an infinite $\sqsupset$-decreasing sequence, which is not possible since $\sqsupset$ is well-founded. If T_j is not a sort, then $(t_i)_{i\geq j}$ is an infinite $\succ_\tau$-decreasing sequence since $\sqsupset$ only compares terms of base type. But this is not possible since $T \geq T_j$ and, by assumption, $[\![T_j]\!]$ satisfies (comp-sn). □

We now show that $\triangleright_a$ preserves computability, and that the structural term ordering $\triangleright^X_{@}$ is stable by computable substitutions of domain X.

Lemma 7.20. *If t is computable and $t \trianglerighteq_a u$, then u is computable.*

Proof. By induction on the definition of $\trianglerighteq_a$. If $t = u$, this is immediate. Otherwise, there are $\mathsf{f}^{\alpha(\mathsf{f})} : \vec{T} \to \mathsf{A}$, $\vec{t} : \vec{T}$ and $i \in \mathrm{Acc}(\mathsf{f})$ such that $t = \mathsf{f}(t_1, \ldots, t_{\alpha(\mathsf{f})})t_{\alpha(\mathsf{f})+1} \ldots t_{|\vec{T}|}$ and $t_i \trianglerighteq_a u$. Since t is computable, by definition of $[\![\mathsf{A}]\!]$, t_i is computable. Therefore, by induction hypothesis, u is computable. □

Lemma 7.21. *If $t : \mathsf{A}$ is computable, $t \triangleright_a u : U$ and A occurs in U, then there is $\mathfrak{b}$ such that $\mathrm{rk}_{\mathsf{A}}(t) = \mathfrak{b} + 1$ and $u \in [\![U]\!]_J$, where $J(\mathsf{A}) = F^{\mathfrak{b}}_{\mathsf{A}}(\varnothing)$ and $J(\mathsf{B}) = I(\mathsf{B})$ if $\mathsf{B} \neq \mathsf{A}$.*

Proof. First note that, by Corollary 7.7, $\mathrm{Sort}_{\leq \mathsf{A}}(U)$ and $\mathrm{Pos}(\mathsf{A}, U) \subseteq \mathrm{Pos}^+(U)$. We now proceed by induction on the definition of $\triangleright_a$. Assume that there are $\mathsf{f}^{\alpha(\mathsf{f})} : \vec{T} \to \mathsf{A}$, $\vec{t} : \vec{T}$ and $i \in \mathrm{Acc}(\mathsf{f})$ such that $t = \mathsf{f}(t_1, \ldots, t_{\alpha(\mathsf{f})})t_{\alpha(\mathsf{f})+1} \ldots t_{|\vec{T}|}$ and $t_i \trianglerighteq_a u$. By definition, $\mathrm{rk}_{\mathsf{A}}(t)$ can be neither 0 nor a limit ordinal. Therefore, there must be $\mathfrak{b}$ such that $\mathrm{rk}_{\mathsf{A}}(t) = \mathfrak{b} + 1$ and $t_i \in [\![T_i]\!]_J$. If $t_i = u$, then we are done. Assume now that $t_i \triangleright_a u$. Then, there is B such that $t_i : \mathsf{B}$. By definition of Acc, $\mathsf{B} \trianglelefteq \mathsf{A}$. By Corollary 7.7, $\mathrm{Sort}_{\trianglelefteq \mathsf{B}}$. Since A occurs in U, we have $\mathsf{A} \trianglelefteq \mathsf{B}$ and thus $\mathsf{B} = \mathsf{A}$ because $\trianglerighteq$ is well-founded by (typ-sn). Hence, by induction hypothesis, there is $\mathfrak{c}$ such that $\mathrm{rk}_{\mathsf{A}}(t_i) = \mathfrak{c} + 1$ and $u \in [\![U]\!]_K$, where $K(\mathsf{A}) = F^{\mathfrak{c}}_{\mathsf{A}}(\varnothing)$ and $K(\mathsf{B}) = I(\mathsf{A})$ if $\mathsf{B} \neq \mathsf{A}$. Therefore, $u \in [\![U]\!]_J$ by Lemma 7.14 since $\mathfrak{c} \leq \mathfrak{b}$, $\mathrm{Sort}_{\trianglelefteq \mathsf{A}}(U)$ and $\mathrm{Pos}(\mathsf{A}, U) \subseteq \mathrm{Pos}^+(U)$. □

Lemma 7.22. *If $t \triangleright_@^X u$, σ is computable on X and $t\sigma$ is computable, then $u\sigma$ is computable and $t\sigma \supseteq u\sigma$.*

Proof. Since $t \triangleright_@^X u$, there are A, v and $\vec{x} : \vec{W}$ such that $t : \mathsf{A}$, $u : \mathsf{A}$, $u = v\vec{x}$, $t \triangleright_a v$, $\vec{x} \in X$ and $\mathrm{Pos}(\mathsf{A}, \vec{W}) = \varnothing$. Therefore, $u\sigma = (v\sigma)(\vec{x}\sigma)$ and, since $\triangleright_a$ is stable by substitution, $t\sigma \triangleright_a v\sigma$. By Lemma 7.21, there is $\mathfrak{b}$ such that $\mathrm{rk}_{\mathsf{A}}(t\sigma) = \mathfrak{b}+1$ and $v\sigma \in [\![\vec{W} \to A]\!]_J$, where $J(\mathsf{A}) = F_{\mathsf{A}}^{\mathfrak{b}}(\varnothing)$ and $J(\mathsf{B}) = I(\mathsf{A})$ if $\mathsf{B} \neq \mathsf{A}$. Since σ is computable on X, we have $\vec{x}\sigma$ computable. Since $\mathrm{Pos}(\mathsf{A}, \vec{W}) = \varnothing$, by Lemma 6.8, we have $\vec{x}\sigma \in [\![\vec{W}]\!]_J$. Therefore, $u\sigma \in [\![\mathsf{A}]\!]_J$ and $t\sigma \supseteq u\sigma$. □

7.6. **Well-foundedness of CPO with accessible subterms.** We now check that type interpretations are computability predicates, and that function symbols are computable.

One can easily check that all the lemmas of Section 6.3 are still valid, as well as the lemmas 6.23 and 6.24 (since they do not depend on $(\mathcal{F}_b\_)$ rules). Therefore, following the proof of Theorem 6.25, we get:

Theorem 7.23. *For all type T, $[\![T]\!]$ is a computability predicate,* i.e. *satisfies* (comp-sn)*,* (comp-red)*,* (comp-neutral) *and* (comp-lam)*.*

Lemma 7.24. *If A is a basic sort, then $[\![\mathsf{A}]\!] = \mathrm{SN}_{\mathsf{A}}(\succ_\tau)$.*

Proof. By Lemma 6.24, it suffices to prove that, for all $t \in \mathrm{SN}_{\mathsf{A}}(\succ_\tau)$, we have $t \in [\![\mathsf{A}]\!]$. By (typ-sn), $\gtrdot$ is well-founded. By Lemma 7.11, $\succ_\tau$ is monotone and thus $\succ_\tau \cup \triangleright$ is well-founded on $\mathrm{SN}(\succ_\tau)$. We can therefore proceed by induction on (A, t) with $(\gtrdot, \succ_\tau \cup \triangleright)$ as well-founded relation.

We first prove that every $\succ_\tau$-reduct u of t is computable. Since $t \in \mathrm{SN}_{\mathsf{A}}(\succ_\tau)$, we have $u \in \mathrm{SN}_U(\succ_\tau)$. By definition of $\succ_\tau$, $\mathsf{A} \geq U$. Therefore, U is a basic sort and, by induction hypothesis, $u \in [\![U]\!]$ since $\mathsf{A} \gtrdot U$ or else $\mathsf{A} = U$ and $t \succ_\tau u$.

Hence, if t is neutral, then $t \in [\![\mathsf{A}]\!]$ since $[\![\mathsf{A}]\!]$ satisfies (comp-neutral). Otherwise, $t = \mathsf{f}(t_1, \dots, t_{\alpha(\mathsf{f})}) t_{\alpha(\mathsf{f})+1} \dots t_{|\vec{T}|}$ with $\mathsf{f}^{\alpha(\mathsf{f})} : \vec{T} \to \mathsf{A}$ and $\mathrm{Acc}(\mathsf{f}) \neq \varnothing$. Let $i \in \mathrm{Acc}(\mathsf{f})$. Then, $\mathrm{Sort}_{\trianglelefteq \mathsf{A}}(T_i)$. Since A is basic, $T_i = \mathsf{A}$ or T_i is a basic sort. In both cases, $T_i \trianglelefteq \mathsf{A}$ and T_i is a basic sort. Therefore, $t_i \in [\![T_i]\!]$ since $t_i \in \mathrm{SN}_{T_i}(\succ_\tau)$ and $\mathsf{A} \gtrdot T_i$ or else $\mathsf{A} = T_i$ and $t \triangleright t_i$. □

Lemma 7.25. *Let $\mathsf{f}^{|\vec{T}|} : \vec{T} \to U$ and $\vec{t} \in [\![\vec{T}]\!]$. Then, $\mathsf{f}(\vec{t})$ is computable.*

Proof. There are $\vec{U}$ and A such that $U = \vec{U} \to \mathsf{A}$. By definition, $\mathsf{f}(\vec{t})$ is computable if, for every $\vec{u} \in [\![\vec{U}]\!]$, $\mathsf{f}(\vec{t})\vec{u}$ is computable. By Theorem 7.23, $[\![\vec{T}]\!]$ and $[\![\vec{U}]\!]$ satisfy (comp-sn) and, by Lemma 7.19, $[\![\vec{T}]\!] \subseteq \mathrm{SN}(\succ_\tau \cup \supseteq)$. Therefore, by Lemma 6.12, $(\succ_{\mathcal{F}}, (\succ_\tau \cup \supseteq)_{\mathrm{stat}})_{\mathrm{lex}}$ is well-founded. We can therefore prove that, for all $((\mathsf{f}, \vec{t}), \vec{u})$ such that $\mathsf{f}(\vec{t})\vec{u}$ is of base type, $\mathsf{f}(\vec{t})\vec{u}$ is computable, by induction on $((\succ_{\mathcal{F}}, (\succ_\tau \cup \supseteq)_{\mathrm{stat}})_{\mathrm{lex}}, (\succ_\tau)_{\mathrm{lex}})_{\mathrm{lex}}$ (0).

Since $\mathsf{f}(\vec{t})\vec{u}$ is of base type and all its accessible arguments are computable by assumption, it suffices to prove that all its $\succ_\tau$-reducts are computable. To this end, we prove that, for all $k \leq n = |\vec{u}|$, every $\succ_\tau$-reduct of $\mathsf{f}(\vec{t})u_1 \dots u_k$ is computable, by induction on k (1).

- $k = 0$. The proof is the same as for Lemma 6.26 except for the new cases:
  - $(\mathcal{F}_b \triangleright)$ There are i, $u : U$ and $v : V$ such that $t_i \trianglerighteq_b^s u \trianglerighteq_a v \succeq_\tau w$. By stability by substitution of $\trianglerighteq_b^s$, $\trianglerighteq_a$ and $\succeq_\tau$ (Lemma 7.9 and 7.11), we have $t_i = t_i\sigma \trianglerighteq_b^s u\sigma \trianglerighteq_a v\sigma \succeq_\tau w\sigma$. Since $\vec{t}$ are computable and $[\![\vec{T}]\!]$ satisfies (comp-sn), we have $\vec{t} \in \mathrm{SN}(\succ_\tau)$. Since $\succ_\tau$ is monotone (Lemma 7.11), we have $u\sigma \in \mathrm{SN}(\succ_\tau)$. Hence, by Lemma 7.24, $u\sigma$ is computable and, by Lemma 7.20, $v\sigma$ is computable. Therefore, $w\sigma$ is computable since, by Theorem 7.23, $[\![V]\!]$ satisfies (comp-red).

– ($\mathcal{F}_b$=) There are g and $\vec{u}$ such that $w = \mathsf{g}(\vec{u})$, $\mathsf{f} \simeq_{\mathcal{F}} \mathsf{g}$, $\vec{t}\,(\succ_\tau \cup \rhd^X_{@} \succeq_\tau)_{\mathrm{stat}(\mathsf{f})}\,\vec{u}$ and $\mathsf{f}(\vec{t}) \succ^X \vec{u}$. Since $\mathsf{f}(\vec{t}) \succ^X \vec{u}$, by induction hypothesis (2), $\vec{u}\sigma$ are computable. If $t_i \succ_\tau u_j$ then, by stability by substitution (Lemma 7.11), $t_i = t_i\sigma \succ_\tau u_j\sigma$. If $t_i \rhd^X_{@} v \succeq_\tau u_j$ then, by Lemma 7.22, $t_i\sigma \sqsupset v\sigma$ and, by stability by substitution again, $v\sigma \succeq_\tau u_j\sigma$. Therefore, by Lemma 7.18 and transitivity, $t_i\sigma \sqsupset u_j\sigma$. Thus, in both cases, $\vec{t}\,(\succ_\tau \cup \sqsupset)_{\mathrm{stat}(\mathsf{f})}\,\vec{u}\sigma$ and, by induction hypothesis (0), $\mathsf{g}(\vec{u})\sigma$ is computable.

- $k > 0$. Then, $\mathsf{f}(\vec{t})\vec{u} = tu_k$ where $t = \mathsf{f}(\vec{t})u_1 \dots u_{k-1}$. By induction hypothesis (1), every $\succ_\tau$-reduct of t is computable. Now, if $u_k \succ_\tau u'_k$, then tu'_k is computable by induction hypothesis (0). Therefore, by Lemma 6.15, every $\succ_\tau$-reduct of tu_k is computable. □

Theorem 7.26. *The relation $\succ_\tau$ of Definition 7.10 is well-founded.*

Proof. After Theorem 6.11, Theorem 7.23 and Lemma 7.25. □

7.7. **Using semantic comparisons.** The extension of CPO described here is still not able to orient the terminating rules defining the recursor of the type C in Example 5.2:

Example 7.27. Given an arbitrary type A, the recursor (of arity 3) at type A of the type C of continuations of Example 5.2 has type $\mathsf{rec}^A_{\mathsf{C}} : \mathsf{C} \to A \to (\neg\neg C \to \neg\neg A \to A) \to A$. Its rewrite rules are the following:

$$\mathsf{rec}^A_{\mathsf{C}}(\mathsf{d}, u, v) \to u$$
$$\mathsf{rec}^A_{\mathsf{C}}(\mathsf{c}(x), u, v) \to v\, x\, (\lambda y^{\neg A} x\, (\lambda z^{\mathsf{C}} y\, \mathsf{rec}^A_{\mathsf{C}}(z, u, v)))$$

The problem is that we do not have $\mathsf{c}(x) \rhd^{\{y,z\}}_{@} z$. Indeed, C is *non-strictly* positive and the structural term ordering can only handle *strictly* positive types.

To handle such rules, we know two solutions. The first one is to define the interpretation of C so that $\mathsf{rec}^A_{\mathsf{C}}$ is computable by definition [92, 71, 11], which is possible since positivity conditions are satisfied. However, this solution lacks flexibility for the user who is forced to define all other functions on C via the recursor.

The second, flexible solution consists in considering types with size annotations (to be interpreted by ordinals) and, in ($\mathcal{F}_b$=), compare terms by their size annotations, an approach initiated independently in [53, 43] and later developed in various works, *e.g.* [1, 8, 9, 19]. Indeed, assuming that $\mathsf{c}(x)$ has type $\mathsf{C}^{\alpha+1}$, then x has type $\neg\neg C^\alpha$ and, thus, the bound variable z gets the type C^α which size annotation is smaller than the one of $\mathsf{c}(x)$.

Including semantics in RPO was pioneered by Kamin and Lévy [60], and extended to HORPO in [22]. In both cases, semantics was added by replacing the precedence by a semantic order on terms. The use of size annotations is a different way to include semantics in these orders. These two different ways of including semantics in recursive path orders are however related: both can be seen as an instance of the more general semantic labeling schema [93, 48, 20].

8. Small symbols

In this section, we consider a further extension of CPO that originated from some draft version of [55] and try to answer the following general question: can we relax the constraints on the precedence? More precisely, to which extent can a function symbol be smaller than

an application or an abstraction? We are going to show that this is indeed possible if the rules governing these *small* symbols are more restrictive than the ones for *big* symbols.

We first define the extension of CPO to small symbols, and then show the computability properties including a specific one for small symbols. Unlike before, this will reveal a circularity among the dependencies between the different computability properties, hence strong normalization does not follow. Breaking this circularity will require assumptions on the types of small symbols that are then investigated for practical purposes. It will appear that, for instance, any constructor of a strictly-positive inductive type can be considered a small symbol.

8.1. **CPO with small symbols.**

Definition 8.1 (CPO with small symbols)**.** We assume that the set of function symbols is partitioned into a set $\mathcal{F}_b$ of *big* symbols and a set $\mathcal{F}_s$ of *small* symbols so that:

- no small symbol is greater or equivalent to a big symbol **(small-lt-big)**
- small symbols with arrow output type have no accessible argument **(small-acc)**

We then extend $\succ^X$ by adding the rules of Figure 3.

We will add conditions on the types of small symbols after Definition 8.9 (see Figure 4).

Figure 3: Additional CPO rules for small symbols

$(@\mathcal{F}_s)$	$tu \succ^X \mathsf{f}(\vec{v})$ if $\mathsf{f} \in \mathcal{F}_s$ and $(\forall i)\ tu \succ^X_\tau v_i$
$(\lambda\mathcal{F}_s)$	$\lambda x t \succ^X \mathsf{f}(\vec{v})$ if $\mathsf{f} \in \mathcal{F}_s$ and $(\forall i)\ \lambda x t \succ^X_\tau v_i$
$(\mathcal{F}_s\triangleright)$	$\mathsf{f}(\vec{t}) \succ^X v$ if $\mathsf{f} \in \mathcal{F}_s$ and $(\exists i)\ t_i \succeq_\tau v$
$(\mathcal{F}_s=)$	$\mathsf{f}(\vec{t}) \succ^X \mathsf{g}(\vec{u})$ if $\mathsf{f} \in \mathcal{F}_s$, $\mathsf{f} \simeq_\mathcal{F} \mathsf{g}$, $(\forall i)\ \mathsf{f}(\vec{t}) \succ^X_\tau u_i$ and $\vec{t}\ (\succ_\tau \cup \triangleright^X_@ \succeq_\tau)_{\mathrm{stat}(\mathsf{f})}\ \vec{u}$
$(\mathcal{F}_s>)$	$\mathsf{f}(\vec{t}) \succ^X \mathsf{g}(\vec{u})$ if $\mathsf{f} \in \mathcal{F}_s$, $\mathsf{f} \succ_\mathcal{F} \mathsf{g}$ and $(\forall i)\ \mathsf{f}(\vec{t}) \succ^X_\tau u_i$
$(\mathcal{F}_s@)$	$\mathsf{f}(\vec{t}) \succ^X u\,v$ if $\mathsf{f} \in \mathcal{F}_s$, $\mathsf{f}(\vec{t}) \succ^X_\tau u$ and $\mathsf{f}(\vec{t}) \succ^X_\tau v$
$(\mathcal{F}_s\mathcal{X})$	$\mathsf{f}(\vec{t}) \succ^X x$ if $\mathsf{f} \in \mathcal{F}_s$ and $x \in X$

Because of the rules $(@\mathcal{F}_s)$ and $(\mathcal{F}_s@)$, one may think that the relation is not terminating anymore, but this is not the case for typing reasons. Indeed, in contrast with rules for big symbols, rules for small symbols require type checking the recursive calls systematically.

For instance, assume that $\mathsf{f} : o \to o$ and $\mathsf{g}^2 : o \to o \to o$. Then, although we have $\mathsf{f}\,\mathsf{a} \succ_\tau \mathsf{g}(\mathsf{a},\mathsf{a})$ by $(@\mathcal{F}_s)$ since $\mathsf{f}\,\mathsf{a} \succ_\tau \mathsf{a}$ by $(@\triangleright)$, we do not hopefully have $\mathsf{g}(\mathsf{a},\mathsf{a}) \succ_\tau \mathsf{f}\,\mathsf{a}$ by $(\mathcal{F}_s@)$ because we do not have $\mathsf{g}(\mathsf{a},\mathsf{a}) \succ_\tau \mathsf{f}$ for typing reasons.

On the other hand, there is no rule $(\mathcal{F}_s\lambda)$ such that $\mathsf{f}(\vec{t}) \succ^X \lambda y v$ if $\mathsf{f}(\vec{t}) \succ^X v$ and $y \notin \mathrm{FV}(v)$ because, together with the rule $(\lambda\mathcal{F}_s)$, it would lead to non-termination as shown by the following example: given small symbols $\mathsf{a} : o \to o \succ_\mathcal{F} \mathsf{b} : o$, $\lambda x \mathsf{b} \succ_\tau \mathsf{a}$ by $(\lambda\mathcal{F}_s)$, and $\mathsf{a} \succ_\tau \lambda x \mathsf{b}$ by $(\mathcal{F}_s\lambda)$ since $\mathsf{a} \succ_\tau \mathsf{b}$ by $(\mathcal{F}_s>)$. It is however possible to have $(\mathcal{F}_s\lambda)$ if one removes $(\lambda\mathcal{F}_s)$. We choose to present the case of $(\mathcal{F}_s\lambda)$ because it seems more useful, but the termination proof can be easily adapted if $(\mathcal{F}_s\lambda)$ is replaced by $(\lambda\mathcal{F}_s)$. Note however that this does not lead to the same definition for the sets SPos, LPos, ... (Definition 8.9) studied in Section 8.4.

Two potential improvements are left. First, take a rule $(\mathcal{F}_s\triangleright)$ similar to the rule $(\mathcal{F}_b\triangleright)$ of Figure 2. Second, get rid of the assumption (small-acc), if possible.

8.2. **Computability properties.**

Lemma 8.2 (Basic properties)**.**

- $\succ^X$ *is well-defined.*
- $\succ_\tau$ *is monotone.*
- *If* $a \succ^X b$*, then* $\mathrm{FV}(b) \subseteq \mathrm{FV}(a) \cup X$*.*
- $\succ^X$ *is stable by* α*-equivalence.*
- $\succ^X$ *is stable by substitution away from* X*.*
- *If* $e, e' \in \mathcal{X}$*,* $\tau(e) = \tau(e')$*,* $t \succ^X u$ *and* $e' \notin \mathrm{FV}(\lambda e u)$*, then* $t \succ^{X-\{e\}\cup\{e'\}} u_e^{e'}$*.*

Keeping the same definitions for neutral terms and the base type interpretation as in Section 6, it is easy to check that Lemma 6.13 and Lemma 6.14 still hold. However, because of the new rules $(@\mathcal{F}_s)$ and $(\lambda\mathcal{F}_s)$, Corollary 6.17 and Corollary 6.19, hence Lemma 6.16 and Lemma 6.18 reveal new dependencies that require introducing the following new computability property for a set S of terms of type T:

(comp-small) $\mathsf{f}(\vec{t}) \in S$ if $\mathsf{f}(\vec{t}) : T$, $\mathsf{f} \in \mathcal{F}_s$ and $\vec{t}$ are computable.

Note that big symbols do not need any computability property because they are bigger than everybody else, and therefore other computability properties do not depend upon the computability of big symbols. It follows that they cannot be implied in any circularity.

Lemma 8.3. *Let* $t : U \to V$ *and* $u : U$*. Then, every* $\succ_\tau$*-reduct of* tu *is computable if:*

- *every* $\succ_\tau$*-reduct of* t *is computable;*
- u *is computable;*
- *if* $t = \lambda x v$*, then* v_x^u *is computable;*
- *for all* u' *such that* $u \succ_\tau u'$*,* tu' *is computable;*
- $[\![U]\!]$ *satisfies* (comp-red)*;*
- $[\![V]\!]$ *satisfies* (comp-red)*;*
- $[\![V']\!]$ *satisfies* (comp-lam) *and* (comp-small) *whenever* $V' \leq V$*.*

Proof. The proof is the same as for Lemma 6.15 except for the new case:

- $(@\mathcal{F}_s)$ $w = \mathsf{f}(\vec{v})$, $\mathsf{f} \in \mathcal{F}_s$ and $(\forall i)$ $tu \succ_\tau v_i$. By induction hypothesis, $\vec{v}$ are computable. Since $[\![W]\!]$ satisfies (comp-small) by assumption, w is computable. □

Lemma 8.4. *Let* $t : U \to V$ *and* $u : U$*. Then,* tu *is computable if:*

- u *is computable;*
- *every* $\succ_\tau$*-reduct of* t *is computable;*
- *if* $t = \lambda x v$*, then* v_x^u *is computable;*
- *either* t *is neutral or* $t = \lambda x v$*;*
- $[\![U]\!]$ *satisfies* (comp-red) *and* (comp-sn)*;*
- $[\![V]\!]$ *satisfies* (comp-red) *and* (comp-neutral)*;*
- $[\![V']\!]$ *satisfies* (comp-lam) *and* (comp-small) *whenever* $V' \leq V$*.*

Proof. As for Lemma 6.16 but using Lemma 8.3 instead. □

Corollary 8.5. $[\![U \to V]\!]$ *satisfies* (comp-neutral) *if:*

- $[\![U]\!]$ *satisfies* (comp-sn) *and* (comp-red)*;*
- $[\![V]\!]$ *satisfies* (comp-red) *and* (comp-neutral)*;*

- $[\![V']\!]$ *satisfies* (comp-lam) *and* (comp-small) *whenever* $V' \leq V$.

Proof. As for Corollary 6.17 but using Lemma 8.4 instead. □

Lemma 8.6. *Let* $x : U$ *and* $v : V$. *Then,* λxv *is computable if:*

- *for all computable* $u : U$, v_x^u *is computable;*
- $[\![U]\!]$ *satisfies* (comp-sn) *and* (comp-red) *and contains a variable, which is the case if it satisfies* (comp-neutral) *too;*
- $[\![V]\!]$ *satisfies* (comp-sn), (comp-red) *and* (comp-neutral)*;*
- $[\![V']\!]$ *satisfies* (comp-lam) *whenever* $V' \leq V$*;*
- $[\![W]\!]$ *satisfies* (comp-small) *whenever* $W \leq U \to V$.

Proof. The proof is the same as for Lemma 6.18 except for the new case:

- $(\lambda\mathcal{F}_s)$ $w = \mathsf{f}(\vec{v})$, $\mathsf{f} \in \mathcal{F}_s$ and $(\forall i)$ $\lambda xv \succ_\tau v_i$. By induction hypothesis, $\vec{v}$ are computable. Thus, w is computable since, by assumption, $[\![W]\!]$ satisfies (comp-small). □

Corollary 8.7. $[\![U \to V]\!]$ *satisfies* (comp-lam) *if:*

- $[\![U]\!]$ *satisfies* (comp-sn), (comp-red) *and* (comp-neutral)*;*
- $[\![V]\!]$ *satisfies* (comp-sn), (comp-red) *and* (comp-neutral)*;*
- $[\![V']\!]$ *satisfies* (comp-lam) *whenever* $V' \leq V$*;*
- $[\![W]\!]$ *satisfies* (comp-small) *whenever* $W \leq U \to V$.

We are left with the new computability property for small symbols:

Lemma 8.8. $[\![U]\!]$ *satisfies* (comp-small) *if:*

- $[\![U]\!]$ *satisfies* (comp-neutral)*;*
- $[\![U']\!]$ *satisfies* (comp-small) *whenever* $U' < U$*;*
- *for every small* $\mathsf{f}^{|\vec{T}|} : \vec{T} \to U$, $[\![\vec{T}]\!]$ *satisfies* (comp-sn) *and* (comp-red).

Proof. Assume that $\mathsf{f}^{\alpha(\mathsf{f})} : \vec{T} \to U$ is small. By assumption, $[\![\vec{T}]\!]$ satisfies (comp-sn) and, by Lemma 7.19, $[\![\vec{T}]\!] \subseteq \mathrm{SN}(\succ_\tau \cup \sqsupset)$. Therefore, by Lemma 6.12, $(\succ_\mathcal{F}, (\succ_\tau \cup \sqsupset)_{\mathrm{stat}})_{\mathrm{lex}}$ is well-founded when restricted to small symbols. We can therefore prove that, for all $(\mathsf{f}, \vec{t}) \in \Sigma$ with $\mathsf{f} \in \mathcal{F}_s$, $\mathsf{f}(\vec{t})$ is computable, by induction on $(\succ_\mathcal{F}, (\succ_\tau \cup \sqsupset)_{\mathrm{stat}})_{\mathrm{lex}}$ (1).

We first prove that $\mathsf{f}(\vec{t})$ is computable if all its $\succ_\tau$-reducts so are. If U is a sort, then the result holds since $\vec{t}$ are computable. Otherwise, by (small-acc), $\mathrm{Acc}(\mathsf{f}) = \varnothing$ and $\mathsf{f}(\vec{t})$ is neutral. Therefore, the result holds since $[\![U]\!]$ satisfies (comp-neutral) by assumption.

We now prove that every $\succ_\tau$-reduct $w : W$ of $\mathsf{f}(\vec{t})$ is computable by induction on w (2). By definition of $\succ_\tau$, we have $U \geq W$.

- $(\mathcal{F}_s\triangleright)$ $(\exists i)$ $t_i \succeq_\tau w$. By assumption, $[\![\vec{T}]\!]$ satisfies (comp-red). Thus, w is computable.
- $(\mathcal{F}_s=)$ There are $\mathsf{g} : \vec{U} \to W$ and $\vec{u} : \vec{U}$ such that $w = \mathsf{g}(\vec{u})$, $(\forall i)$ $\mathsf{f}(\vec{t}) \succ_\tau u_i$, $\mathsf{f} \simeq_\mathcal{F} \mathsf{g}$ and $\vec{t}\,(\succ_\tau \cup \triangleright_@^\varnothing \succeq_\tau)_{\mathrm{stat}(\mathsf{f})}\,\vec{u}$. By (small-lt-big), g is small. Since $\mathsf{f}(\vec{t}) \succ_\tau \vec{u}$, by induction hypothesis (2), $\vec{u}$ are computable. We distinguish two cases:
  - $U > W$. Then, $\mathsf{g}(\vec{u})$ is computable since $[\![W]\!]$ satisfies (comp-small).
  - $U = W$. If $t_i \triangleright_@^\varnothing v \succeq_\tau u_j$ then, by Lemma 7.22, $t_i \sqsupset v$ and, by Lemma 7.18, $v \sqsupseteq u_j$. Therefore, by transitivity, $t_i \sqsupset u_j$. Hence, $\vec{t}\,(\succ_\tau \cup \sqsupset)_{\mathrm{stat}(\mathsf{f})}\,\vec{u}$ and, by induction hypothesis (1), $\mathsf{g}(\vec{u})$ is computable.
- $(\mathcal{F}_s>)$ There are $\mathsf{g} : \vec{U} \to W$ and $\vec{u} : \vec{U}$ such that $w = \mathsf{g}(\vec{u})$, $(\forall i)$ $\mathsf{f}(\vec{t}) \succ_\tau u_i$ and $\mathsf{f} >_\mathcal{F} \mathsf{g}$. By (small-lt-big), g is small. Since $\mathsf{f}(\vec{t}) \succ_\tau \vec{u}$, by induction hypothesis (2), $\vec{u}$ are computable. We distinguish two cases:

  - $U > W$. Then, $\mathsf{g}(\vec{u})$ is computable since $[\![W]\!]$ satisfies (comp-small).
  - $U = W$. Then, $\mathsf{g}(\vec{u})$ is computable by induction hypothesis (1).
- ($\mathcal{F}_s$@) There are u and v such that $w = uv$ and $\mathsf{f}(\vec{t}) \succ_\tau uv$. By induction hypothesis (2), u and v are computable. Therefore, uv is computable.
- ($\mathcal{F}_s\mathcal{X}$) Not possible. □

8.3. **Well-foundedness of CPO with small symbols.** In contrast with the previous cases, we cannot conclude from the above lemmas that, for every type T, $[\![T]\!]$ is a computability predicate, because of circularities.

Indeed, for $[\![\mathsf{A}]\!]$ to satisfy (comp-small), we need, for every small symbol $\mathsf{f} : T \to \mathsf{A}$, $[\![T]\!]$ to satisfy (comp-sn); but for $[\![T]\!]$ to satisfy (comp-sn) when $T = U \to V$, we need $[\![U]\!]$ to satisfy (comp-neutral); but for $[\![U]\!]$ to satisfy (comp-neutral) when $U = W \to \mathsf{A}$, we need $[\![\mathsf{A}]\!]$ to satisfy (comp-small). To break this circularity, we will make these dependencies more precise by introducing sets of positions in types that reflect how these computability properties depend from each other. The idea here is that if there is no problematic occurrence of A in T, then $[\![T]\!]$ satisfies (comp-sn), and similarly for the other properties.

Instead of sets of positions, we could have simply considered boolean functions returning true if T contains a problematic occurrence of A. Considering positions allows us to pinpoint precisely which occurrences are problematic, and therefore to obtain sharper conditions on $\mathcal{F}_s$ ensuring the absence of cycle in the dependency graph of the computability properties. Of course, one may think that there are different ways to carry out these proofs, resulting in different dependency graphs. We believe that these relationship are intrinsic to the computability properties, although we have not been able to substantiate this claim so far.

Definition 8.9 (Computability-property positions)**.** For each computability property (S standing for (comp-sn), R for (comp-red), N for (comp-neutral), L for (comp-lam) and C for (comp-small)), we inductively define a set of positions in a type T wrt a sort A as follows:

- $\mathrm{CPos}_\mathsf{A}(\mathsf{A}) = \{\varepsilon\}$ and $\mathrm{CPos}_\mathsf{A}(\mathsf{B}) = \varnothing$ if $\mathsf{B} \neq \mathsf{A}$
- $\mathrm{SPos}_\mathsf{A}(\mathsf{B}) = \mathrm{RPos}_\mathsf{A}(\mathsf{B}) = \mathrm{NPos}_\mathsf{A}(\mathsf{B}) = \mathrm{LPos}_\mathsf{A}(\mathsf{B}) = \varnothing$ whatever A and B are
- $\mathrm{CPos}_\mathsf{A}(U \to V) = \mathrm{NPos}_\mathsf{A}(U \to V)$
- $\mathrm{SPos}_\mathsf{A}(U \to V) = \mathrm{RPos}_\mathsf{A}(U \to V) = \{1p \mid p \in \mathrm{NPos}_\mathsf{A}(U)\} \cup \{2p \mid p \in \mathrm{SPos}_\mathsf{A}(V)\}$
- $\mathrm{NPos}_\mathsf{A}(U \to V) = \{1p \mid p \in \mathrm{SPos}_\mathsf{A}(U) \cup \mathrm{RPos}_\mathsf{A}(U)\}$
  $\cup \{2p \mid p \in \mathrm{RPos}_\mathsf{A}(V) \cup \mathrm{NPos}_\mathsf{A}(V) \cup \mathrm{LPos}_\mathsf{A}(V) \cup \mathrm{CPos}_\mathsf{A}(V)\}$
- $\mathrm{LPos}_\mathsf{A}(U \to V) = \mathrm{CPos}_\mathsf{A}(U \to V)$
  $\cup \{1p \mid p \in \mathrm{SPos}_\mathsf{A}(U) \cup \mathrm{RPos}_\mathsf{A}(U) \cup \mathrm{NPos}_\mathsf{A}(U)\}$
  $\cup \{2p \mid p \in \mathrm{SPos}_\mathsf{A}(V) \cup \mathrm{RPos}_\mathsf{A}(V) \cup \mathrm{NPos}_\mathsf{A}(V) \cup \mathrm{LPos}_\mathsf{A}(V) \cup \mathrm{CPos}_\mathsf{A}(V)\}$

Note that $\mathrm{RPos}_\mathsf{A}(T) = \mathrm{SPos}_\mathsf{A}(T) \subseteq \mathrm{LPos}_\mathsf{A}(T)$ and $\mathrm{NPos}_\mathsf{A}(T) \subseteq \mathrm{CPos}_\mathsf{A}(T)$. Straightforward simplifications then yield:

- $\mathrm{NPos}_\mathsf{A}(U \to V) = \{1p \mid p \in \mathrm{SPos}_\mathsf{A}(U)\} \cup \{2p \mid p \in \mathrm{LPos}_\mathsf{A}(V) \cup CPos_\mathsf{A}(V)\}$
- $\mathrm{LPos}_\mathsf{A}(U \to V)$
  $= \mathrm{CPos}_\mathsf{A}(U \to V) \cup \{1p \mid p \in \mathrm{SPos}_\mathsf{A}(U) \cup \mathrm{NPos}_\mathsf{A}(U)\} \cup \{2p \mid p \in \mathrm{LPos}_\mathsf{A}(V) \cup CPos_\mathsf{A}(V)\}$

We can now express in Figure 4 conditions on the types of the small symbols ensuring, as we shall show next, the absence of cycles in the dependency graph.

Consider the (small-sort) case and assume that $T_i \trianglelefteq \mathsf{A}$. Then, either $T_i = \mathsf{A}$ and $\mathrm{SPos}_\mathsf{A}(T_i) = \varnothing$ by definition, or $T_i \triangleleft \mathsf{A}$ and $\mathrm{SPos}_\mathsf{A}(T_i) = \varnothing$ by Lemma 8.11. The condition for base types is therefore (strictly) weaker than the one for arrow types. This weaker

Figure 4: Conditions on types of small symbols

$\forall \mathsf{f}^{\lvert\vec{T}\rvert} : \vec{T} \to \mathsf{A}, (\forall i)\ \mathrm{Sort}_{\leqslant \mathsf{A}}(T_i) \wedge \mathrm{SPos}_{\mathsf{A}}(T_i) = \varnothing$	**(small-sort)**
$\forall \mathsf{f}^{\lvert\vec{T}\rvert} : \vec{T} \to \vec{U} \to \mathsf{A}$ with $\lvert\vec{U}\rvert > 0$, $\mathrm{Acc}(\mathsf{f}) = \varnothing \wedge (\forall i)\ \mathrm{Sort}_{\leqslant \mathsf{A}}(T_i) \wedge T_i \leqslant \vec{U} \to \mathsf{A}$	**(small-arrow)**

form will indeed be important later for deciding if a function symbol of base output type can be declared small.

Lemma 8.10. *If* $\mathrm{Sort}_{\leqslant \mathsf{A}}(T)$*, then* $\mathrm{SPos}_{\mathsf{A}}(T) = \mathrm{NPos}_{\mathsf{A}}(T) = \mathrm{LPos}_{\mathsf{A}}(T) = \mathrm{CPos}_{\mathsf{A}}(T) = \varnothing$*.*

Proof. By induction on T.

- $T = \mathsf{B}$. Then, $\mathrm{SPos}_{\mathsf{A}}(T) = \mathrm{NPos}_{\mathsf{A}}(T) = \mathrm{LPos}_{\mathsf{A}}(T) = \varnothing$ by definition. Since $\mathrm{Sort}_{\leqslant \mathsf{A}}(T)$, we have $\mathsf{B} \neq \mathsf{A}$ and thus $\mathrm{CPos}_{\mathsf{A}}(T) = \varnothing$ too.
- $T = U \to V$. Since $\mathrm{Sort}_{< \mathsf{A}}(U)$ and $\mathrm{Sort}_{< \mathsf{A}}(V)$, $\mathrm{SPos}_{\mathsf{A}}(U) = \mathrm{NPos}_{\mathsf{A}}(U) = \mathrm{LPos}_{\mathsf{A}}(U) = \mathrm{CPos}_{\mathsf{A}}(U) = \varnothing$ and $\mathrm{SPos}_{\mathsf{A}}(V) = \mathrm{NPos}_{\mathsf{A}}(V) = \mathrm{LPos}_{\mathsf{A}}(V) = \mathrm{CPos}_{\mathsf{A}}(V) = \varnothing$ by induction hypothesis. Thus, $\mathrm{SPos}_{\mathsf{A}}(T) = \mathrm{NPos}_{\mathsf{A}}(T) = \mathrm{LPos}_{\mathsf{A}}(T) = \mathrm{CPos}_{\mathsf{A}}(T) = \varnothing$. □

Lemma 8.11. *If* $T > T'$ *and* $\mathrm{Sort}_{\leqslant \mathsf{A}}(T)$ *then:*

- $\mathrm{SPos}_{\mathsf{A}}(T') = \varnothing$ *whenever* $\mathrm{SPos}_{\mathsf{A}}(T) = \varnothing$*,*
- $\mathrm{NPos}_{\mathsf{A}}(T') = \varnothing$ *whenever* $\mathrm{NPos}_{\mathsf{A}}(T) = \varnothing$*,*
- $\mathrm{LPos}_{\mathsf{A}}(T') = \varnothing$ *whenever* $\mathrm{LPos}_{\mathsf{A}}(T) = \varnothing$*,*
- $\mathrm{CPos}_{\mathsf{A}}(T') = \varnothing$ *whenever* $\mathrm{CPos}_{\mathsf{A}}(T) = \varnothing$*.*

Proof. We proceed by induction on T. Note that $\mathrm{Sort}_{\leqslant \mathsf{A}}(T')$ by Lemma 2.6.

- $T = \mathsf{B}$. Since $\mathrm{Sort}_{\leqslant \mathsf{A}}(T)$, we have $\mathsf{B} \leq \mathsf{A}$. By transitivity, $T' < \mathsf{A}$. Hence, by Lemma 2.5, $\mathrm{Sort}_{\leqslant \mathsf{A}}(T')$. Therefore, $\mathrm{SPos}_{\mathsf{A}}(T') = \mathrm{NPos}_{\mathsf{A}}(T') = \mathrm{LPos}_{\mathsf{A}}(T') = \mathrm{CPos}_{\mathsf{A}}(T') = \varnothing$ by Lemma 8.10.
- $T = U \to V$. Then, $\mathrm{Sort}_{\leqslant \mathsf{A}}(U)$ and $\mathrm{Sort}_{\leqslant \mathsf{A}}(V)$.
  - $\mathrm{SPos}_{\mathsf{A}}(T) = \varnothing$. Then, $\mathrm{NPos}_{\mathsf{A}}(U) = \varnothing$ and $\mathrm{SPos}_{\mathsf{A}}(V) = \varnothing$. By (typ-arrow), there are two cases:
    - $V \geq T'$. Then, $\mathrm{SPos}_{\mathsf{A}}(T') = \varnothing$ by induction hypothesis.
    - $T' = U \to V'$ and $V > V'$. By induction hypothesis, $\mathrm{SPos}_{\mathsf{A}}(V') = \varnothing$. Therefore, $\mathrm{SPos}_{\mathsf{A}}(T') = \varnothing$.
  - $\mathrm{CPos}_{\mathsf{A}}(T) = \mathrm{NPos}_{\mathsf{A}}(T) = \varnothing$. Then, $\mathrm{SPos}_{\mathsf{A}}(U) = \varnothing$ and $\mathrm{SPos}_{\mathsf{A}}(V) = \mathrm{NPos}_{\mathsf{A}}(V) = \mathrm{LPos}_{\mathsf{A}}(V) = \mathrm{CPos}_{\mathsf{A}}(V) = \varnothing$. By (typ-arrow), there are two cases:
    - $V \geq T'$. Then, $\mathrm{CPos}_{\mathsf{A}}(T') = \mathrm{NPos}_{\mathsf{A}}(T') = \varnothing$ by induction hypothesis.
    - $T' = U \to V'$ and $V > V'$, hence $\mathrm{SPos}_{\mathsf{A}}(V') = \mathrm{NPos}_{\mathsf{A}}(V') = \mathrm{LPos}_{\mathsf{A}}(V') = \mathrm{CPos}_{\mathsf{A}}(V') = \varnothing$ by induction hypothesis. $\mathrm{CPos}_{\mathsf{A}}(T') = \mathrm{NPos}_{\mathsf{A}}(T') = \varnothing$ follows.
  - $\mathrm{LPos}_{\mathsf{A}}(T) = \varnothing$. Then, $\mathrm{SPos}_{\mathsf{A}}(U) = \mathrm{NPos}_{\mathsf{A}}(U) = \varnothing$ and $\mathrm{SPos}_{\mathsf{A}}(V) = \mathrm{NPos}_{\mathsf{A}}(V) = \mathrm{LPos}_{\mathsf{A}}(V) = \mathrm{CPos}_{\mathsf{A}}(V) = \varnothing$. By (typ-arrow), there are two cases:
    - $V \geq T'$. Then, $\mathrm{LPos}_{\mathsf{A}}(T') = \varnothing$ by induction hypothesis.
    - $T' = U \to V'$ and $V > V'$. By induction hypothesis, $\mathrm{SPos}_{\mathsf{A}}(V) = \mathrm{NPos}_{\mathsf{A}}(V) = \mathrm{LPos}_{\mathsf{A}}(V) = \mathrm{CPos}_{\mathsf{A}}(V) = \varnothing$. Therefore, $\mathrm{LPos}_{\mathsf{A}}(T') = \varnothing$.

Lemma 8.12. *Assume that the condition (small-arrow) of Figure 4 holds. Let* A *be a sort such that, for all sort* $\mathsf{B} \lessdot \mathsf{A}$*,* $[\![\mathsf{B}]\!]$ *satisfies* (comp-small)*, and let* T *be a type such that* $\mathrm{Sort}_{\leqslant \mathsf{A}}(T)$*. Then:*

- $[\![T]\!]$ *satisfies* (comp-sn) *and* (comp-red) *if* $\mathrm{SPos}_{\mathsf{A}}(T) = \varnothing$*,*

- $[\![T]\!]$ *satisfies* (comp-neutral) *if* $\mathrm{NPos}_{\mathsf{A}}(T) = \varnothing$,
- $[\![T]\!]$ *satisfies* (comp-lam) *if* $\mathrm{LPos}_{\mathsf{A}}(T) = \varnothing$,
- $[\![T]\!]$ *satisfies* (comp-small) *if* $\mathrm{CPos}_{\mathsf{A}}(T) = \varnothing$.

Proof. We proceed by induction on $\gtrdot$ which is well-founded by (typ-sn).

- $T = \mathsf{B}$. Since $\mathrm{Sort}_{\leqslant\mathsf{A}}(T)$, we have $\mathsf{B} \leqslant \mathsf{A}$.
  - $\mathrm{SPos}_{\mathsf{A}}(T) = \varnothing$. $[\![T]\!]$ satisfies (comp-red) by Lemma 6.23. By Lemma 6.24, $[\![T]\!]$ satisfies (comp-sn) if, for all $U < T$, $[\![U]\!]$ satisfies (comp-sn). So, let $U < T$. By transitivity, $U \lessdot \mathsf{A}$. Hence, by Lemma 2.5, $\mathrm{Sort}_{\lessdot\mathsf{A}}(U)$ and, by Lemma 8.10, $\mathrm{SPos}_{\mathsf{A}}(U) = \varnothing$. Therefore, by induction hypothesis, $[\![U]\!]$ satisfies (comp-sn).
  - $\mathrm{NPos}_{\mathsf{A}}(T) = \varnothing$. $[\![T]\!]$ satisfies (comp-neutral) by Lemma 6.23.
  - $\mathrm{LPos}_{\mathsf{A}}(T) = \varnothing$. $[\![T]\!]$ satisfies (comp-lam) by Lemma 6.23.
  - $\mathrm{CPos}_{\mathsf{A}}(T) = \varnothing$. Then, $\mathsf{B} \lessdot \mathsf{A}$ and, by assumption, $[\![T]\!]$ satisfies (comp-small).
- $T = U \to V$. Then, $\mathrm{Sort}_{\leqslant\mathsf{A}}(U)$ and $\mathrm{Sort}_{\leqslant\mathsf{A}}(V)$.
  - $\mathrm{SPos}_{\mathsf{A}}(T) = \varnothing$. Then, $\mathrm{NPos}_{\mathsf{A}}(U) = \varnothing$ and $\mathrm{SPos}_{\mathsf{A}}(V) = \varnothing$. By induction hypothesis, $[\![U]\!]$ satisfies (comp-neutral) and $[\![V]\!]$ satisfies (comp-sn) and (comp-red). Hence, $[\![T]\!]$ satisfies (comp-sn) and (comp-red) by Lemmas 6.13 and 6.14.
  - $\mathrm{NPos}_{\mathsf{A}}(T) = \varnothing$. Then, $\mathrm{SPos}_{\mathsf{A}}(U) = \varnothing$ and $\mathrm{SPos}_{\mathsf{A}}(V) = \mathrm{NPos}_{\mathsf{A}}(V) = \mathrm{LPos}_{\mathsf{A}}(V) = \mathrm{CPos}_{\mathsf{A}}(V) = \varnothing$. By induction hypothesis, $[\![U]\!]$ satisfies (comp-sn) and (comp-red). Let now $V' \leq V$. By Lemma 2.6, $\mathrm{Sort}_{\leqslant\mathsf{A}}(V')$. By Lemma 8.11, $\mathrm{SPos}_{\mathsf{A}}(V') = \mathrm{NPos}_{\mathsf{A}}(V') = \mathrm{LPos}_{\mathsf{A}}(V') = \mathrm{CPos}_{\mathsf{A}}(V') = \varnothing$. By (typ-right-subterm) and transitivity, $T > V'$. Hence, by induction hypothesis, $[\![V']\!]$ satisfies (comp-red), (comp-neutral), (comp-lam) and (comp-small). Therefore, by Corollary 8.5, $[\![T]\!]$ satisfies (comp-neutral).
  - $\mathrm{CPos}_{\mathsf{A}}(T) = \varnothing$. Then, $\mathrm{NPos}_{\mathsf{A}}(T) = \varnothing$, hence $\mathrm{SPos}_{\mathsf{A}}(U) = \varnothing$ and $\mathrm{SPos}_{\mathsf{A}}(V) = \mathrm{NPos}_{\mathsf{A}}(V) = \mathrm{LPos}_{\mathsf{A}}(V) = \mathrm{CPos}_{\mathsf{A}}(V) = \varnothing$. We now check the conditions of Lemma 8.8:
    - $[\![T]\!]$ satisfies (comp-neutral) since $\mathrm{NPos}_{\mathsf{A}}(T) = \varnothing$.
    - Let $W < T$. We prove that $\mathrm{CPos}_{\mathsf{A}}(W) = \varnothing$. By (typ-arrow), there are two cases:
      - $V \geq W$. Then, by Lemma 8.11, $\mathrm{CPos}_{\mathsf{A}}(W) = \varnothing$.
      - $W = U \to V'$ and $V > V'$. Then, by Lemma 8.11, $\mathrm{SPos}_{\mathsf{A}}(V') = \mathrm{NPos}_{\mathsf{A}}(V') = \mathrm{LPos}_{\mathsf{A}}(V') = \mathrm{CPos}_{\mathsf{A}}(V') = \varnothing$. Thus, $\mathrm{CPos}_{\mathsf{A}}(W) = \varnothing$.

      Hence, by induction hypothesis, $[\![W]\!]$ satisfies (comp-small).
    - Let now $\mathsf{f}^{|\vec{T}|} : \vec{T} \to T$ be small. There are $\vec{B}$ and B such that $V = \vec{V} \to \mathsf{B}$. So, by (small-arrow), $(\forall i)\ \mathrm{Sort}_{\leqslant\mathsf{B}}(T_i)$ and $T_i \leqslant T$.
      We first prove that, if $\mathrm{Sort}_{\leqslant\mathsf{A}}(\vec{S} \to \mathsf{B})$ and $\mathrm{CPos}_{\mathsf{A}}(\vec{S} \to \mathsf{B}) = \varnothing$, then $\mathsf{B} \lessdot \mathsf{A}$, by induction on $\vec{S}$. If $\vec{S}$ is empty, then $\mathrm{Sort}_{\leqslant\mathsf{A}}(\mathsf{B})$ and $\mathrm{CPos}_{\mathsf{A}}(\mathsf{B}) = \varnothing$. Thus, $\mathsf{B} \lessdot \mathsf{A}$. If $\vec{S} = U\vec{V}$, then $\mathrm{CPos}_{\mathsf{A}}(\vec{S} \to \mathsf{B}) = \varnothing$ implies that $\mathrm{CPos}_{\mathsf{A}}(\vec{V} \to \mathsf{B}) = \varnothing$. Hence, by induction hypothesis, $\mathsf{B} \lessdot \mathsf{A}$.
      We therefore have $\mathsf{B} \lessdot \mathsf{A}$ for $T = U\vec{V} \to \mathsf{B}$, $\mathrm{Sort}_{\leqslant\mathsf{A}}(T)$ and $\mathrm{CPos}_{\mathsf{A}}(T) = \varnothing$. Hence, $\mathrm{Sort}_{\lessdot\mathsf{A}}(\vec{T})$ and, by Lemma 8.10, $\mathrm{SPos}_{\mathsf{A}}(\vec{T}) = \varnothing$. If $T_i \lessdot T$, then $[\![T_i]\!]$ satisfies (comp-red) and (comp-sn) by induction hypothesis. Otherwise, $T_i = T$ and $[\![T_i]\!]$ satisfies (comp-red) and (comp-sn) as shown previously.
  - $\mathrm{LPos}_{\mathsf{A}}(T) = \varnothing$. Then, $\mathrm{SPos}_{\mathsf{A}}(U) = \mathrm{NPos}_{\mathsf{A}}(U) = \varnothing$ and $\mathrm{SPos}_{\mathsf{A}}(V) = \mathrm{NPos}_{\mathsf{A}}(V) = \mathrm{LPos}_{\mathsf{A}}(V) = \mathrm{CPos}_{\mathsf{A}}(V) = \varnothing$. By induction hypothesis, $[\![U]\!]$ satisfies (comp-sn), (comp-red) and (comp-neutral).

Let now $V' \leq V$. By Lemma 2.6, $\mathrm{Sort}_{\leqslant \mathsf{A}}(V')$. By Lemma 8.11, $\mathrm{SPos}_{\mathsf{A}}(V') = \mathrm{NPos}_{\mathsf{A}}(V') = \mathrm{LPos}_{\mathsf{A}}(V') = \mathrm{CPos}_{\mathsf{A}}(V') = \varnothing$. By (typ-right-subterm) and transitivity, $T > V'$. Hence, by induction hypothesis, $[\![V']\!]$ satisfies (comp-sn), (comp-red), (comp-neutral), (comp-lam) and (comp-small).
Let now $W \leq T$. By Lemma 2.6, $\mathrm{Sort}_{\leqslant \mathsf{A}}(W)$. Since $\mathrm{LPos}_{\mathsf{A}}(T) = \varnothing$, we have $\mathrm{CPos}_{\mathsf{A}}(T) = \varnothing$. Hence $\mathrm{CPos}_{\mathsf{A}}(W) = \varnothing$ by Lemma 8.11. If $W = T$, we have already seen that $[\![T]\!]$ satisfies (comp-small). Otherwise, $W > T$ and, by induction hypothesis, $[\![W]\!]$ satisfies (comp-small).
Therefore, by Corollary 8.7, $[\![T]\!]$ satisfies (comp-lam). □

Theorem 8.13. *Assume that the conditions of Figure 4 hold. For all type T, $[\![T]\!]$ is a computability predicate,* i.e. *satisfies* (comp-sn)*,* (comp-red)*,* (comp-neutral)*,* (comp-lam) *and* (comp-small)*.*

Proof. We proceed by induction on $\gtrdot$ which is well-founded by assumption (typ-sn). We distinguish two cases:

- T is a sort A. By Lemma 6.23, $[\![\mathsf{A}]\!]$ satisfies (comp-red), (comp-neutral), (comp-lam). By Lemma 6.24 and induction hypothesis, $[\![\mathsf{A}]\!]$ satisfies (comp-sn).
  Let $W < \mathsf{A}$. By Lemma 2.5, $\mathrm{Sort}_{\leqslant \mathsf{A}}(W)$. By Lemma 8.10, $\mathrm{CPos}_{\mathsf{A}}(W) = \varnothing$. Therefore, $[\![W]\!]$ satisfies (comp-small) by Lemma 8.12.
  Let now $\mathsf{f}^{\alpha(\mathsf{f})} : \vec{T} \to \mathsf{A}$ be small. By (small-sort), we have $(\forall i)\ \mathrm{Sort}_{\leqslant \mathsf{A}}(T_i)$ and $\mathrm{SPos}_{\mathsf{A}}(T_i) = \varnothing$. Therefore, $[\![\vec{T}]\!]$ satisfies (comp-sn) and (comp-red) by Lemma 8.12. Hence, $[\![\mathsf{A}]\!]$ satisfies (comp-small) by Lemma 8.8.
- Otherwise, $T = U \to V$. Since $T \rhd_l U$, by induction hypothesis, $[\![U]\!]$ is a computability predicate. Let now V' be a type such that $V \geq V'$. By (typ-right-subterm) and transitivity, $T > V'$. By induction hypothesis, $[\![V']\!]$ is a computability predicate. Therefore, $[\![U \to V]\!]$ satisfies (comp-sn) by Lemma 6.13, (comp-red) by Lemma 6.14, (comp-neutral) by Corollary 8.5, (comp-small) by Lemma 8.8 and (comp-lam) by Corollary 8.7. □

Theorem 8.14. *If the conditions of Figure 4 hold, then the relation $\succ_\tau$ of Definition 8.1 is well-founded.*

Proof. After Theorem 6.11, Theorem 8.13 and Lemma 7.25. □

8.4. **Checking computability assumptions for small symbols.** We explore here simple sufficient conditions under which the set $\mathrm{SPos}_{\mathsf{A}}(T)$ is empty, and therefore, which symbols whose output type is a sort A can be declared as small. The order of a type plays an important role here. In case these conditions are not met, it is of course always possible to check (small-sort) and (small-arrow), which are both decidable.

Lemma 8.15. $\mathrm{SPos}_{\mathsf{A}}(T) = \varnothing$ *if* $o(T) \leq 1$.

Proof. We proceed by induction on T.

- $T = \mathsf{B}$. Then, $\mathrm{SPos}_{\mathsf{A}}(T) = \varnothing$ by definition.
- $T = U \to V$. Since $o(T) \leq 1$, $o(U) \leq 0$ and $o(V) \leq 1$. U being a sort, $\mathrm{NPos}_{\mathsf{A}}(U) = \varnothing$. Since $o(V) \leq 1$, $\mathrm{SPos}_{\mathsf{A}}(V) = \varnothing$ by induction hypothesis. Hence $\mathrm{SPos}_{\mathsf{A}}(T) = \varnothing$. □

Can therefore be declared as small, any symbol whose type is of order less than or equal to 2 since its arguments have then a type of order less than or equal to 1. This is in particular the case of the constructors of first-order data types.

More generally, can be declared as small every constructor of a strictly-positive inductive type, whatever its order is, which is the class of inductive types allowed in the Coq proof assistant [54]:

Lemma 8.16. *Given types* $\vec{T}$ *and a sort* A, $\mathrm{SPos}_{\mathsf{A}}(\vec{T} \to \mathsf{A}) = \varnothing$ *if* $\mathrm{Sort}_{\lessdot\mathsf{A}}(\vec{T})$.

Proof. By induction on T.

- $T = \mathsf{A}$. Immediate.
- $T = U \to V$. Then, $\mathrm{Sort}_{\lessdot\mathsf{A}}(U)$ and V is of the form $\vec{T} \to \mathsf{A}$ with $\mathrm{Sort}_{\lessdot\mathsf{A}}(\vec{T})$. By Lemma 8.10, $\mathrm{NPos}_{\mathsf{A}}(U) = \varnothing$. By induction hypothesis, $\mathrm{SPos}_{\mathsf{A}}(V) = \varnothing$. Therefore, $\mathrm{SPos}_{\mathsf{A}}(T) = \varnothing$. □

Non-strictly positive types are not available in Coq because strong elimination rules may cause non-terminating computations in Coq's richer type system [29]. Nothing such that can happen in our simple type system in which constructors of non-strictly positive inductive types of order ≤ 2 can be declared as small:

Lemma 8.17. $\mathrm{NPos}_{\mathsf{A}}(T) = \mathrm{LPos}_{\mathsf{A}}(T) = \mathrm{CPos}_{\mathsf{A}}(T) = \varnothing$ *if* $o(T) \leq 1$, $\mathrm{Sort}_{\leqslant\mathsf{A}}(T)$ *and* $\mathrm{Pos}(\mathsf{A}, T) \subseteq \mathrm{Pos}^{-}(T)$.

Proof. We proceed by induction on T.

- $T = \mathsf{B}$. Then, $\mathrm{NPos}_{\mathsf{A}}(T) = \mathrm{LPos}_{\mathsf{A}}(T) = \varnothing$ by definition. Since $\mathrm{Sort}_{\leqslant\mathsf{A}}(T)$, we have $\mathsf{B} \leq \mathsf{A}$. Since $\mathrm{Pos}(\mathsf{A}, T) \subseteq \mathrm{Pos}^{-}(T)$ and $\mathrm{Pos}^{-}(T) = \varnothing$, we have $\mathsf{B} \neq \mathsf{A}$. Therefore, $\mathrm{CPos}_{\mathsf{A}}(T) = \varnothing$.
- $T = U \to V$. Since $o(T) \leq 1$, we have $o(U) \leq 0$ and $o(V) \leq 1$. Thus, U is a sort and $\mathrm{SPos}_{\mathsf{A}}(U) = \mathrm{NPos}_{\mathsf{A}}(U) = \varnothing$. By Lemma 8.15, $\mathrm{SPos}_{\mathsf{A}}(V) = \varnothing$. Since $\mathrm{Sort}_{\leqslant\mathsf{A}}(T)$, we have $\mathrm{Sort}_{\leqslant\mathsf{A}}(V)$. Since $\mathrm{Pos}(\mathsf{A}, T) \subseteq \mathrm{Pos}^{-}(T)$, we have $\mathrm{Pos}(\mathsf{A}, V) \subseteq \mathrm{Pos}^{-}(V)$. Hence, by induction hypothesis, $\mathrm{NPos}_{\mathsf{A}}(V) = \mathrm{LPos}_{\mathsf{A}}(V) = \mathrm{CPos}_{\mathsf{A}}(V) = \varnothing$. Therefore, $\mathrm{NPos}_{\mathsf{A}}(T) = \mathrm{LPos}_{\mathsf{A}}(T) = \mathrm{CPos}_{\mathsf{A}}(T) = \varnothing$. □

Lemma 8.18. $\mathrm{SPos}_{\mathsf{A}}(T) = \varnothing$ *if* $o(T) \leq 2$, $\mathrm{Sort}_{\leqslant\mathsf{A}}(T)$ *and* $\mathrm{Pos}(\mathsf{A}, T) \subseteq \mathrm{Pos}^{+}(T)$.

Proof. We proceed by induction on T.

- $T = \mathsf{B}$. Then, $\mathrm{SPos}_{\mathsf{A}}(T) = \varnothing$ by definition.
- $T = U \to V$. Since $o(T) \leq 2$, we have $o(U) \leq 1$ and $o(V) \leq 2$. Since $\mathrm{Sort}_{\leqslant\mathsf{A}}(T)$, we have $\mathrm{Sort}_{\leqslant\mathsf{A}}(U)$ and $\mathrm{Sort}_{\leqslant\mathsf{A}}(V)$. Since $\mathrm{Pos}(\mathsf{A}, T) \subseteq \mathrm{Pos}^{+}(T)$, we have $\mathrm{Pos}(\mathsf{A}, U) \subseteq \mathrm{Pos}^{-}(U)$ and $\mathrm{Pos}(\mathsf{A}, V) \subseteq \mathrm{Pos}^{+}(V)$. Hence, by Lemma 8.17, $\mathrm{NPos}_{\mathsf{A}}(U) = \varnothing$ and, by induction hypothesis, $\mathrm{SPos}_{\mathsf{A}}(V) = \varnothing$. Therefore, $\mathrm{SPos}_{\mathsf{A}}(T) = \varnothing$. □

But positivity is not always sufficient as shown by the following example. Assume that $\mathsf{f} : T \to \mathsf{A}$ with $T = (\mathsf{B} \to N) \to \mathsf{A}$, $N = (\mathsf{B} \to \mathsf{A}) \to \mathsf{B}$ and $\mathsf{B} < \mathsf{A}$. The sort A occurs negatively in N and positively in T, which is a 3rd order type. We cannot declare f as small since we do not know how to prove that $[\![T]\!]$ satisfies (comp-sn) by using our lemmas. Indeed, to prove that $[\![T]\!]$ satisfies (comp-sn), we need to prove that $[\![\mathsf{B} \to N]\!]$ satisfies (comp-neutral) (Lemma 6.13). To prove that $[\![\mathsf{B} \to N]\!]$ satisfies (comp-neutral), we need to prove that $[\![N]\!]$ satisfies (comp-lam) (Corollary 8.5). To prove that $[\![N]\!]$ satisfies (comp-lam), we need to prove that $[\![\mathsf{B} \to \mathsf{A}]\!]$ satisfies (comp-neutral) (Corollary 8.7). To prove that $[\![\mathsf{B} \to \mathsf{A}]\!]$ satisfies (comp-neutral), we need to prove that $[\![\mathsf{A}]\!]$ satisfies (comp-small) (Corollary 8.5). But, to

prove that $[\![\mathsf{A}]\!]$ satisfies (comp-small), we need to prove that $[\![T]\!]$ satisfies (comp-sn) (Lemma 8.8). The circularity has not been broken here, but we can of course declare f as being big instead of small.

8.5. **Examples.** In this section, we analyze two examples that show the need for small symbols and their use. We will see that CPO with small symbols contains not only core CPO, but also a subset of its transitive closure. But CPO with small symbols is not transitive either, as shown by the second example which needs the use of both small symbols and the transitive closure.

Example 8.19. Taken from the Termination Problems Data Base (TPDB) [87] under the name `Applicative_05__TreeFlatten`. Let a be a sort. Consider the function symbols $\mathsf{nil} : \mathsf{a}$, $\mathsf{flatten} : \mathsf{a} \to \mathsf{a}$, $\mathsf{concat}^1 : \mathsf{a} \to \mathsf{a}$, $\mathsf{cons}^2 : \mathsf{a} \to \mathsf{a} \to \mathsf{a}$, $\mathsf{append}^2 : \mathsf{a} \to \mathsf{a} \to \mathsf{a}$, $\mathsf{node}^2 : \mathsf{a} \to \mathsf{a} \to \mathsf{a}$ and $\mathsf{map}^2 : (\mathsf{a} \to \mathsf{a}) \to \mathsf{a} \to \mathsf{a}$.

The higher-order rewrite system

$$\begin{aligned}
\mathsf{map}(F, \mathsf{nil}) &\to \mathsf{nil}\\
\mathsf{map}(F, \mathsf{cons}(x, v)) &\to \mathsf{cons}(F\ x, \mathsf{map}(F, v))\\
\mathsf{flatten}\ \mathsf{node}(x, v) &\to \mathsf{cons}(x, \mathsf{concat}(\mathsf{map}(\mathsf{flatten}, v)))\\
\mathsf{concat}(\mathsf{nil}) &\to \mathsf{nil}\\
\mathsf{concat}(\mathsf{cons}(x, v)) &\to \mathsf{append}(x, \mathsf{concat}(v))\\
\mathsf{append}(\mathsf{nil}, v) &\to v\\
\mathsf{append}(\mathsf{cons}(x, u), v) &\to \mathsf{cons}(x, \mathsf{append}(u, v))
\end{aligned}$$

can be proved terminating with CPO by considering concat, append, map, cons and nil small, while node and $\mathsf{flatten}$ can be either small or big (we consider them as big in the following). All symbols can have multiset status. Let the precedence be $\mathsf{concat} >_{\mathcal{F}} \mathsf{append} >_{\mathcal{F}} \mathsf{cons}$, $\mathsf{node} >_{\mathcal{F}} \mathsf{map} >_{\mathcal{F}} \mathsf{nil}$, $\mathsf{node} >_{\mathcal{F}} \mathsf{flatten}$ and $\mathsf{map} >_{\mathcal{F}} \mathsf{cons}$.

Let us show the proof of the third rule, which is the most interesting one. Since cons is small, we apply first $(@\mathcal{F}_s)$ and then we recursively need $\mathsf{flatten}\ \mathsf{node}(x, v) >_\tau x$, which holds by $(@\triangleright)$ and then $(\mathcal{F}_b\triangleright)$, and $\mathsf{flatten}\ \mathsf{node}(x, v) >_\tau \mathsf{concat}(\mathsf{map}(\mathsf{flatten}, v))$, which needs $(@\mathcal{F}_s)$ again. We then recursively need $\mathsf{flatten}\ \mathsf{node}(x, v) >_\tau \mathsf{map}(\mathsf{flatten}, v)$, which generates the subgoal $\mathsf{node}(x, v) >_\tau \mathsf{map}(\mathsf{flatten}, v)$ by $(@\triangleright)$, and then the new subgoals $\mathsf{node}(x, v) >_\tau \mathsf{flatten}$ and $\mathsf{node}(x, v) >_\tau v$ by $(\mathcal{F}_b>)$. We conclude by $(\mathcal{F}_b>)$ and $(\mathcal{F}_b\triangleright)$.

The above example cannot be shown by core CPO because $\mathsf{flatten}$ is curried and the head symbol of the third rule is then an application. There is however a way out with the transitive closure of core CPO if one allows the introduction of new symbols. Let $\mathsf{flattenunc}^1 : \mathsf{a} \to \mathsf{a}$ be a new symbol. Assuming for example $\mathsf{flatten} >_{\mathcal{F}} \mathsf{flattenunc}$ and $\mathsf{node} >_{\mathcal{F}} \{\mathsf{cons}, \mathsf{concat}, \mathsf{map}, \mathsf{flatten}\}$, we can then show the successive ordering comparisons:

$$\begin{aligned}
\mathsf{flatten}\ \mathsf{node}(x, v) &>_\tau (\lambda x\, \mathsf{flattenunc}(x))\, \mathsf{node}(x, v)\\
(\lambda x\, \mathsf{flattenunc}(x))\, \mathsf{node}(x, v) &>_\tau \mathsf{flattenunc}(\mathsf{node}(x, v))\\
\mathsf{flattenunc}(\mathsf{node}(x, v)) &>_\tau \mathsf{cons}(x, \mathsf{concat}(\mathsf{map}(\mathsf{flatten}, v)))
\end{aligned}$$

The first reduces to $\mathsf{flatten} >_\tau \lambda x\, \mathsf{flattenunc}(x)$, the second is a β-reduction, and the third is a classical RPO-like computation. Details are left to the reader.

The use of small symbols can therefore help showing termination of examples that would otherwise require the use of the transitive closure of core CPO (as well as a signature extension in the above case). Small symbols, however, do not make CPO transitive. Our second example requires indeed using both small symbols *and* the transitive closure:

Example 8.20. Taken from TPDB under the name `AotoYamada_05__014`. Let a and b be sorts. Consider the function symbols $0 : \mathsf{b}$, $\mathsf{nil} : \mathsf{a}$, $\mathsf{inc} : \mathsf{a} \to \mathsf{a}$, $\mathsf{double} : \mathsf{a} \to \mathsf{a}$, $\mathsf{s}^1 : \mathsf{b} \to \mathsf{b}$, $\mathsf{plus}^1 : \mathsf{b} \to \mathsf{b} \to \mathsf{b}$, $\mathsf{times}^1 : \mathsf{b} \to \mathsf{b} \to \mathsf{b}$, $\mathsf{map}^1 : (\mathsf{b} \to \mathsf{b}) \to \mathsf{a} \to \mathsf{a}$, and $\mathsf{cons}^2 : \mathsf{b} \to \mathsf{a} \to \mathsf{a}$.

The higher-order rewrite system

$$\begin{aligned}
\mathsf{plus}(0)\ x &\to x\\
\mathsf{plus}(\mathsf{s}(y))\ x &\to \mathsf{s}(\mathsf{plus}(y)\ x)\\
\mathsf{times}(0)\ x &\to 0\\
\mathsf{times}(\mathsf{s}(y))\ x &\to \mathsf{plus}(\mathsf{times}(y)\ x)\ x\\
\mathsf{map}(F)\ \mathsf{nil} &\to \mathsf{nil}\\
\mathsf{map}(F)\ \mathsf{cons}(x, v) &\to \mathsf{cons}(F\ x, \mathsf{map}(F)\ v))\\
\mathsf{inc} &\to \mathsf{map}(\mathsf{plus}(\mathsf{s}(0)))\\
\mathsf{double} &\to \mathsf{map}(\mathsf{times}(\mathsf{s}(\mathsf{s}(0))))
\end{aligned}$$

can be proved terminating with CPO by taking $\mathsf{a} = \mathsf{b}$ in the type ordering, cons and s as small symbols, the precedence $\mathsf{times} >_{\mathcal{F}} \mathsf{plus}$, $\mathsf{inc} >_{\mathcal{F}} \{\mathsf{map}, \mathsf{plus}, 0\}$, $\mathsf{double} >_{\mathcal{F}} \{\mathsf{map}, \mathsf{times}, 0\}$, and status multiset for all symbols.

We consider the 4th rule, for which we shall use the transitive closure of CPO, and the 6th rule, for which small symbols are needed (for the second rule too).

For the 4th rule, we exhibit the middle term $(\lambda z\, \mathsf{plus}(\mathsf{times}(y)\, z)\, z)\, x$ which is smaller than the lefthand side and β-reduces to the righthand side of the rule.

To prove that $\mathsf{times}(\mathsf{s}(y))\, x$ is greater than this middle term, we apply $(@=)$, and since the second arguments are equal, we have to show that $\mathsf{times}(\mathsf{s}(y)) >_\tau (\lambda z\, \mathsf{plus}(\mathsf{times}(y)\, z)\, z)$. Since, both terms have the same type, by $(\mathcal{F}_b\lambda)$ and then $(\mathcal{F}_b@)$, we are left to show $\mathsf{times}(\mathsf{s}(y)) >^{\{z\}} \mathsf{plus}(\mathsf{times}(y)\, z)$, since $\mathsf{times}(\mathsf{s}(y)) >^{\{z\}} z$ holds by $(\mathcal{F}_b\mathcal{X})$. For this last check, we apply first $(\mathcal{F}_b>)$ and then $(\mathcal{F}_b@)$, since $\mathsf{times}(\mathsf{s}(y)) >^{\{z\}} \mathsf{times}(y)$ holds by $(\mathcal{F}_b=)$ and then $(\mathcal{F}_s\triangleright)$, and $\mathsf{times}(\mathsf{s}(y)) >^{\{z\}} z$ holds by $(\mathcal{F}_b\mathcal{X})$.

For the 6th rule, we apply first $(@\mathcal{F}_s)$, which requires to check $\mathsf{map}(F)\, \mathsf{cons}(x, v) >_\tau F\, x$ and $\mathsf{map}(F)\, \mathsf{cons}(x, v) >_\tau \mathsf{map}(F)\, v$. Since the types of both sides are equivalent, the first one holds by applying $(@=)$ and then $(\mathcal{F}_b\triangleright)$ to the first argument and $(\mathcal{F}_s\triangleright)$ to the second one. Finally, for $\mathsf{map}(F)\, \mathsf{cons}(x, v) >_\tau \mathsf{map}(F)\, v$, we apply $(@=)$ and then $(\mathcal{F}_s\triangleright)$ to the second argument.

9. Conclusion

We have defined in this paper a well-founded relation on algebraic lambda-terms following a type discipline accepting simple types in the sense of Church, and inductive types in the sense of Martin-Löf. Further, we could easily cope with (implicitly) universally quantified type variables as in [59], a type discipline called weak polymorphism.

We want to stress that core CPO has reached a point where we cannot expect any major improvement, as indicated by the counter-examples found to our own attempts to improve it. We are in great debt with Cynthia Kop and Femke van Raamsdonk for igniting this quest, by providing us with an example that removing the type check in the rule $(\mathcal{F}_b=)$ results in losing the well-foundedness property [61]. The very existence of these counter-examples supports our conviction that CPO defines an extremely sharp decidable approximation of sets of rules for which there exists a computability predicate.

Of course, all these counter-examples still hold when adding inductive types and small symbols. We did our best to exploit the idea of small symbols as much as possible within

our proof frame, but cannot argue that the conditions on the signature of small symbols are all necessary and that the corresponding recursive calls cannot be improved: we did not extend our quest for counter-examples to this question. We finally believe that there is also some room left for improving the accessibility relationship, which is restricted so far to terms headed by a function symbol, possibly applied to extra arguments.

A more challenging problem to be investigated now is the generalization of this new definition to the calculus of constructions along the lines of [91] and the suggestions made in [59], where an RPO-like ordering on types was proposed which allowed to give a single definition for terms and types. Generalizing CPO to dependent types appears to follow the classical route initiated in [49], albeit non-trivial [55]. We therefore believe that this work should be applicable to Dedukti [21, 79] with limited effort. On the other hand, we have failed so far to generalize CPO to truly polymorphic types: its use in the proof assistant Coq [54] will require much more effort.

Finally, note that HORPO [65] on the one hand, and the notion of computability closure on the other hand [13], have already been formalized in the proof assistant Coq [54]. These works could serve as a basis for formalizing the results presented in this paper and develop a termination certificate verifier for CPO.

Acknowledgements. The authors thank the reviewers for their suggestions.

References

[1] A. Abel. Termination checking with types. *Theoretical Informatics and Applications*, 38(4):277–319, 2004.

[2] T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.

[3] F. Barbanera. Adding algebraic rewriting to the calculus of constructions: strong normalization preserved. In *Proceedings of the 2nd International Workshop on Conditional and Typed Rewriting Systems*, Lecture Notes in Computer Science 516, 1990.

[4] F. Barbanera and M. Fernández. Modularity of termination and confluence in combinations of rewrite systems with λ_ω. In *Proceedings of the 20th International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science 700, 1993.

[5] F. Barbanera and M. Fernández. Combining first and higher order rewrite systems with type assignment systems. In *Proceedings of the 1st International Conference on Typed Lambda Calculi and Applications*, Lecture Notes in Computer Science 664, 1993.

[6] F. Barbanera, M. Fernández, and H. Geuvers. Modularity of strong normalization and confluence in the algebraic-λ-cube. In *Proceedings of the 9th IEEE Symposium on Logic in Computer Science*, 1994.

[7] H. Barendregt. Lambda calculi with types. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of logic in computer science. Volume 2. Background: computational structures*, pages 117–309. Oxford University Press, 1992.

[8] G. Barthe, M. J. Frade, E. Giménez, L. Pinto, and T. Uustalu. Type-based termination of recursive definitions. *Mathematical Structures in Computer Science*, 14(1):97–141, 2004.

[9] F. Blanqui. A type-based termination criterion for dependently-typed higher-order rewrite systems. In *Proceedings of the 15th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 3091, 2004.

[10] F. Blanqui. Definitions by rewriting in the calculus of constructions. *Mathematical Structures in Computer Science*, 15(1):37–92, 2005.

[11] F. Blanqui. Inductive types in the calculus of algebraic constructions. *Fundamenta Informaticae*, 65(1-2):61–86, 2005.

[12] F. Blanqui. (HO)RPO revisited. Technical Report 5972, INRIA, France, 2006.

[13] F. Blanqui. A formalization in Coq of the notion of computability closure for proving the termination of rewrite relations on 2013.

[14] F. Blanqui. Termination of rewrite relations on λ-terms based on Girard's notion of reducibility. *Theoretical Computer Science*, ?(?):?–?, 2015. To appear.
[15] F. Blanqui, J.-P. Jouannaud, and M. Okada. Inductive-data-type systems. *Theoretical Computer Science*, 272:41–68, 2002.
[16] F. Blanqui, J.-P. Jouannaud, and A. Rubio. Higher-order termination: from Kruskal to computability. In *Proceedings of the 13th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, Lecture Notes in Computer Science 4246, 2006. Invited paper.
[17] F. Blanqui, J.-P. Jouannaud, and A. Rubio. HORPO with computability closure: a reconstruction. In *Proceedings of the 14th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, Lecture Notes in Computer Science 4790, 2007.
[18] F. Blanqui, J.-P. Jouannaud, and A. Rubio. The computability path ordering: the end of a quest. In *Proceedings of the 22nd International Conference on Computer Science Logic*, Lecture Notes in Computer Science 5213, 2008. Invited paper.
[19] F. Blanqui and C. Riba. Combining typing and size constraints for checking the termination of higher-order conditional rewrit In *Proceedings of the 13th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, Lecture Notes in Computer Science 4246, 2006.
[20] F. Blanqui and C. Roux. On the relation between sized-types based termination and semantic labelling. In *Proceedings of the 23rd International Conference on Computer Science Logic*, Lecture Notes in Computer Science 5771, 2009.
[21] M. Boespflug, Q. Carbonneaux, and O. Hermant. The lambda-pi-calculus modulo as a universal proof language. In *Proceedings of the 2nd International Workshop on Proof eXchange for Theorem Proving*, CEUR Workshop Proceedings 878, 2012.
[22] C. Borralleras and A. Rubio. A monotonic higher-order semantic path ordering. In *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, Lecture Notes in Computer Science 2250, 2001.
[23] C. Borralleras and A. Rubio. THOR, a tool for proving the termination of higher-order rewrite systems, 2010.
[24] V. Breazu-Tannen and J. Gallier. Polymorphic rewriting conserves algebraic strong normalization. In *Proceedings of the 16th International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science 372, 1989.
[25] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.
[26] M. Codish, J. Giesl, P. Schneider-Kamp, and R. Thiemann. SAT solving for termination proofs with recursive path orders and dependency pairs. *Journal of Automated Reasoning*, 49(1):53–93, 2011.
[27] E. Contejean, C. Marché, A. P. Tomás, and X. Urbain. Mechanically proving termination using polynomial interpretations. *Journal of Automated Reasoning*, 34(4):325–363, 2005.
[28] T. Coquand. Pattern matching with dependent types. In *Proceedings of the International Workshop on Types for Proofs and Programs*, 1992.
[29] T. Coquand and C. Paulin-Mohring. Inductively defined types. In *Proceedings of the International Conference on Computer Logic*, Lecture Notes in Computer Science 417, 1988.
[30] P. Cousot and R. Cousot. Constructive versions of Tarski's fixed point theorems. *Pacific Journal of Mathematics*, 82(1):43–57, 1979.
[31] H. B. Curry and R. Feys. *Combinatory logic*. North-Holland, 1958.
[32] M. Dauchet. Termination of rewriting is undecidable in the one-rule case. In *Proceedings of the 13th International Symposium on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science 324, 1988.
[33] N. Dershowitz. Orderings for term rewriting systems. In *Proceedings of the 20th IEEE Symposium on Foundations of Computer Science*, 1979.
[34] N. Dershowitz. Orderings for term rewriting systems. *Theoretical Computer Science*, 17:279–301, 1982.
[35] N. Dershowitz. Jumping and escaping: modular termination and the abstract path ordering. *Theoretical Computer Science*, 464:35–47, 2012. Special issue: New Directions in Rewriting (Honoring the 60th Birthday of Yoshihito Toyama).
[36] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science. Volume B: formal models and methods*, chapter 6, pages 243–320. North-Holland, 1990.

[37] N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, 1979.
[38] D. Dougherty. Adding algebraic rewriting to the untyped lambda calculus. *Information and Computation*, 101(2):251–267, 1992.
[39] M. Fernández and J.-P. Jouannaud. Modular termination of term rewriting systems revisited. In *Proceedings of the 10th International Workshop on Specification of Abstract Data Types*, Lecture Notes in Computer Science 906, 1994.
[40] C. Fuhs and C. Kop. Polynomial interpretations for higher-order rewriting. In *Proceedings of the 23rd International Conference on Rewriting Techniques and Applications*, Leibniz International Proceedings in Informatics 15, 2012.
[41] R. O. Gandy. An early proof of normalization by a. m. turing. In J. R. Hindley and J. P. Seldin, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 453–455. Academic Press, 1980.
[42] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006.
[43] E. Giménez. *Un calcul de constructions infinies et son application à la vérification de systèmes communiquants*. PhD thesis, ENS Lyon, France, 1996.
[44] J.-Y. Girard. Une extension de l'interprétation de Gödel à l'analyse et son application à l'élimination des coupures dans l'ana In J. Fenstad, editor, *Proceedings of the 2nd Scandinavian Logic Symposium*, volume 63 of *Studies in Logic and the Foundations of Mathematics*, pages 63–92. North-Holland, 1971.
[45] J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur*. PhD thesis, Université Paris 7, France, 1972.
[46] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and types*. Cambridge University Press, 1988.
[47] J. Goubault-Larrecq. Well-founded recursive relations. In *Proceedings of the 15th International Conference on Computer Science Logic*, Lecture Notes in Computer Science 2142, 2001.
[48] M. Hamana. Higher-order semantic labelling for inductive datatype systems. In *Proceedings of the 9th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*, 2007.
[49] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, 1993.
[50] N. Hirokawa and A. Middeldorp. Tyrolean Termination Tool: techniques and features. *Information and Computation*, 205(4):474–511, 2007.
[51] N. Hirokawa and A. Middeldorp. Uncurrying for termination. In *Proceedings of the 15th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, Lecture Notes in Computer Science 5330, 2008.
[52] M. Hofmann. Approaches to recursive data types - a case study. Unpublished note cited in [72] p. 61, 1995.
[53] J. Hughes, L. Pareto, and A. Sabry. Proving the correctness of reactive systems using sized types. In *Proceedings of the 23th ACM Symposium on Principles of Programming Languages*, 1996.
[54] INRIA, France. *The Coq reference manual, version 8.4pl5*, 2014.
[55] J.-P. Jouannaud and J. Li. Termination of Dependently Typed Rewrite Rules. In *Proceedings of the 13th International Conference on Typed Lambda Calculi and Applications*, Leibniz International Proceedings in Informatics 38, 2015.
[56] J.-P. Jouannaud and M. Okada. A computation model for executable higher-order algebraic specification languages. In *Proceedings of the 6th IEEE Symposium on Logic in Computer Science*, 1991.
[57] J.-P. Jouannaud and M. Okada. Abstract data type systems. *Theoretical Computer Science*, 173(2):349–391, 1997.
[58] J.-P. Jouannaud and A. Rubio. The higher-order recursive path ordering. In *Proceedings of the 14th IEEE Symposium on Logic in Computer Science*, 1999.
[59] J.-P. Jouannaud and A. Rubio. Polymorphic higher-order recursive path orderings. *Journal of the ACM*, 54(1):1–48, 2007.
[60] S. Kamin and J.-J. Lévy. Attempts for generalizing the recursive path orderings. Unpublished note, 1980.
[61] C. Kop. Personal communication, 2008.

[62] C. Kop. Higher order dependency pairs for algebraic functional systems. In *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications*, Leibniz International Proceedings in Informatics 10, 2011.
[63] C. Kop. *Higher order termination*. PhD thesis, VU University Amsterdam, NL, 2012.
[64] C. Kop and F. van Raamsdonk. Higher-order dependency pairs with argument filterings. In *11th International Workshop on Termination*, 2010.
[65] A. Koprowski. Coq formalization of the higher-order recursive path ordering. *Applicable Algebra in Engineering Communication and Computing*, 20(5-6):379–425, 2009.
[66] M. S. Krishnamoorthy and P. Narendran. On recursive path ordering. *Theoretical Computer Science*, 40(2-3):323–328, 1985.
[67] C. Kuratowski. Une méthode d'élimination des nombres transfinis des raisonnements mathématiques. *Fundamenta Mathematicae*, 3(1):76–108, 1922.
[68] K. Kusakari, Y. Isogai, M. Sakai, and F. Blanqui. Static dependency pair method based on strong computability for higher-ord
IEICE Transactions on Information and Systems, E92-D(10):2007–2015, 2009.
[69] D. Lankford. On proving term rewriting systems are Noetherian. Technical report, Lousiana Technical University, USA, 1979.
[70] Z. Manna and S. Ness. On the termination of Markov algorithms. In *Proceedings of the 3rd Hawaii International Conference on System Sciences*, 1970.
[71] R. Matthes. *Extensions of system F by iteration and primitive recursion on monotone inductive types*. PhD thesis, Ludwig Maximilians Universität, München, Germany, 1998.
[72] R. Matthes. *Lambda calculus: a case for inductive definitions*, 2000.
[73] N. P. Mendler. *Inductive definition in type theory*. PhD thesis, Cornell University, USA, 1987.
[74] N. P. Mendler. Inductive types and type constraints in the second-order lambda calculus. *Annals of Pure and Applied Logic*, 51(1-2):159–172, 1991.
[75] P. Narendran, M. Rusinowitch, and R. Verma. RPO constraint solving is in NP. In *Proceedings of the 12th International Conference on Computer Science Logic*, Lecture Notes in Computer Science 1584, 1999.
[76] R. Nieuwenhuis. Simple LPO constraint solving methods. *Information Processing Letters*, 47(2):65–69, 1993.
[77] M. Okada. Strong normalizability for the combined system of the typed lambda calculus and an arbitrary convergent term re
In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, 1989.
[78] C. Roux. *Size-based termination: semantics and generalizations*. PhD thesis, Université Henri Poincaré, Nancy, France, 2011.
[79] R. Saillard. Dedukti 2.3, 2014.
[80] M. Sakai and K. Kusakari. On dependency pair method for proving termination of higher-order rewrite systems. *IEICE Transactions on Information and Systems*, E88-D(3):583–593, 2005.
[81] L. E. Sanchis. Functionals defined by recursion. *Notre Dame Journal of Formal Logic*, 8:161–174, 1967.
[82] S. Suzuki, K. Kusakari, and F. Blanqui. Argument filterings and usable rules in higher-order rewrite systems. *IPSJ Transactions on Programming*, 4(2):1–12, 2011.
[83] W. W. Tait. Intensional interpretations of functionals of finite type I. *Journal of Symbolic Logic*, 32(2):198–212, 1967.
[84] W. W. Tait. A realizability interpretation of the theory of species. In R. Parikh, editor, *Proceedings of the 1972 Logic Colloquium*, volume 453 of *Lecture Notes in Mathematics*, 1975.
[85] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.
[86] TeReSe. *Term rewriting systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.
[87] Termination pproblem data base (TPDB), version 9.0.2, 2014.
[88] A. S. Troelstra. Models and Computability. In A. S. Troelstra, editor, *Metamathematical investigation of intuitionistic arithmetic and analysis*, volume 344 of *Lecture Notes in Mathematics*, pages 97–174. Springer, 1973.
[89] A. M. Turing. Some theorems about Church's system. Unpublished typescript reproduced in [41], 1942.
[90] J. van de Pol. *Termination of higher-order rewrite systems*. PhD thesis, Utrecht Universiteit, NL, 1996.
[91] D. Walukiewicz-Chrząszcz. Termination of rewriting in the calculus of constructions. *Journal of Functional Programming*, 13(2):339–414, 2003.

[92] B. Werner. *Une théorie des constructions inductives*. PhD thesis, Université Paris 7, France, 1994.
[93] H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24:89–105, 1995.